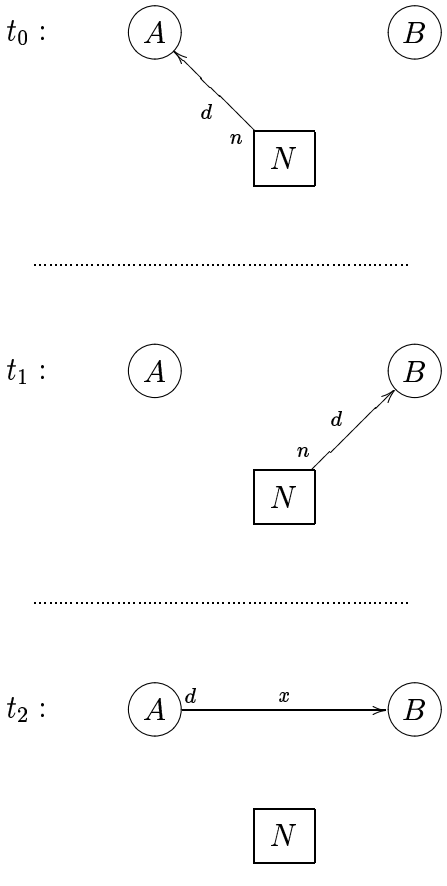
Mowgli

Mobile Work, Gadgets, Language and Infrastructure



968

Jon Haugsand, Shahrzade Mazaher, Anders Moen, Bjarte M. Østvold, Arne-Kristian Groven, Håvard Hegna

Oslo
December 2000

NR Norsk Regnesentral
ANVENDT DATAFORSKNING

Rapport/Report

Tittel/Title: Mowgli:
Mobile Work, Gadgets, Language and Infrastructure

Dato/Date: December
År/Year: 2000
ISBN: 82-539-0474-6
Publikasjonsnr./
Publication no.: 968

Forfatter/Author: Jon Haugsand, Shahrzade Mazaher, Anders Moen, Bjarte M. Østvold, Arne-Kristian Groven, Håvard Hegna

Sammendrag/Abstract: We look at software support for mobile users from two different viewpoints: a framework-based approach aiming at identifying the necessary concepts, and a formal approach using process calculus for rigorous definitions of the concepts. We present a scientific method for analyzing concepts that we call 'conceptual archaeology' and this underlies our work.

The first viewpoint gives the necessary elements in a framework supporting mobile users based on the limitations of mobile environments and the requirements of mobile users. Among the necessary elements is an understanding of the concept session.

The second viewpoint is one of formal analysis. Traditionally π-calculus and related families of theories attempt to describe adequately the domain of distributed systems. We take the tools from a recent textbook by Robin Milner in use to test the adequacy of the π-calculus as a modeling tool in the domain. Among the concept we attempt to model is 'session'.

Emneord/Keywords: conceptual analysis, mobile services, pi-calculus, formal models, sessions

Målgruppe/Target group: Anyone

Tilgjengelighet/Availability: Normal/Public

Prosjekt/Project: Mowgli

Prosjektnr./Project no.: 1021

Satsningsfelt/Research field: Formal methods and languages

Antall sider/No. of pages: 59

Norsk Regnesentral/Norwegian Computing Center
Gaustadalléen 23, Postboks 114 Blindern, 0314 Oslo, Norway
Telefon 22 85 25 00, telefax 22 69 76 60, web http://www.nr.no

Contents

List of figures

Preface

This document is the final report of the Mowgli project, carried out by the authors at the Norwegian Computing Center (NR).

As part of the project a lecture series on process calculi was given at NR in the second half of 2000. The main lecturers of the series were Anders Moen (also organiser) and Bjarte M. Østvold, with other project members filling in. The project's work on approaches to analysis was published on an international workshop [29]; that paper corresponds to Chapter 1 of the report.

The title of this report, corresponding with the project name, indicates a wider focus of research than the end results. A more appropriate title would have been "Mobile systems: infrastructure and formal analysis".

The project's work on the concept session will be used in our future work as a benchmark to decide the fruitfulness of applying π-calculus as a means for doing conceptual analysis in a precise manner, rather than the common imprecise way done in software engineering.

Thematic overview

This report consists of 4 chapters. The first three can be read independently but Chapter 3 must be read before Chapter 4.

Chapter 1 explains the scientific approach or method applied, and gives examples of the various concepts of 'session'. Chapter 2 studies the existing literature on work done to support mobile users and establishes a set of elements necessary in a framework supporting mobile users based on the limitations of mobile environments and the requirements of mobile users. It also defines the concept of session in a general way as to allow mobile users to continue the same task from a different device. Chapter 3 gives a short and self-contained presentation of π-calculus, with a flavor of recursion added to it. Chapter 4 starts the more detailed archaeological analysis, and writes out (specifies) simple contexts of communication.

The report should be considered as the starting point (the first rocket) of a long investigation into a framework supporting mobile users, as Chapter 4 indicates.

Acknowledgements

Naci Akkök, Kjell Åge Bringsrud, Martin Kirkengen, Thor Kristoffersen and Wolfgang Leister participated in discussions about the concept 'session'. Hab-

tamu Abie participated in the seminar on process calculi.

The Mowgli project was financed by NR's *grunnfondsbevilgning*, provided by The Research Council of Norway.

Chapter 1

Conceptual analysis by technical means

1.1 Introduction

Theoretical computer science has traditionally been used to give a scientific, which means mathematical, foundation of new and old programming languages and to study computability in its purity.[1] Following the paradigm of Milner[2], we want to exploit the various formulations of the Pi-calculus and similar systems like the Spi-calculus of [1] and the theory of ambients of Luca Cardelli and Andrew Gordon,[3] in order to do conceptual analysis of presumably well understood concepts in computer science, and especially mobile systems [4]. But it is not obvious that the new theories of theoretical computer science, will contribute to the evolution of mobile distributed systems, either on the conceptual or the technological frontier. Milner writes that:

> "(...) there are many ways of thinking about interactive systems - implies the need to tie these ways together. If a basic set of ideas such as the π-calculus can supply this integrity then designers will respect it, one may dare to hope, in the way that mechanical or electronical engineers respect the differential calculus, which ties together their ways of thinking."[5]

And he continues to reflect about the role his calculus should play;

> "Beyond this conceptual unification, a good outcome for the π-calculus would be to generate new high-level languages and analytical tools, much in the way that its predecessor CCS and CSP contributed to the design of LOTOS, a language designed to express communications protocols."[6]

[1] Complexity theory, subrecursion theory and recursion theory including all their different formulations.

[2] As presented in [26], [25], and [32]

[3] As presented in [6], [7] and [5].

[4] This chapter was published in [29]

[5] [25] p.153.

[6] [25] p.153.

As we can see from these paragraphs, Milner is certainly ambivalent to the reception of his calculus, both humble[7] and ambitious.[8] And he has good reason for this ambivalence. Distributed systems, mobile systems exists independent of the π-calculus. Why should anyone in the computing industry sit down and learn a new complicated theory when there is already so much to learn? Technology works so why bother? We shall indicate that the increasing complexity of the applications we deal with would benefit from formal and conceptual basic research.

In addition to Milner's conceptual unification and the generation of analytic tools for constructing high-level programming languages, we suggest a third approach. We will use the analytic tools for doing *conceptual analysis*, in our version that we shall call *conceptual archaeology*.

1.2 The origins of conceptual analysis

The origin of conceptual analysis is ancient philosophy. In antiquity there was no division between science, theology and philosophy. This meant that scientific investigations were guarded by philosophical reflections, and to some extent, philosophical reflections were influenced by scientific considerations.

1.2.1 Gottlob Frege's contribution

A major turning-point in the history of philosophy and logic, is the work the German mathematician and philosopher Gottlob Frege in his writings in the two last decades of the 19'th century. Frege's contribution to science is twofold. He clears the ground for modern first order logic in [9], [10] and second order logic in [14], as shown in [2], by giving a formal system of first and second order logic and by giving birth to semantics. Frege is also the founder of analytic philosophy, by investigating metaphysical problems within the boundaries of language in [11], [12], [13] and [15], in arguing for the distinction between language, meaning and reference[9], pointing out that they should be divided into three distinct conceptual layers. Analytic philosophy is the commitment to do philosophy by scientific methods. This means to argue and formulate philosophical propositions in such a way that the propositions can be falsified. An analytic philosopher formulates her propositions with the commitment of being true or false.

For Frege conceptual analysis and formal logic are closely interrelated. By using a formal language he has a tool against which the concepts in mathematics and science in general can be calibrated. Frege investigated the laws of thought. The laws of thought could only be unraveled by a pure investigation into the logical laws of the assertorical propositions in a formal language.[10]

[7] "(...) one may dare to hope (...)"
[8] "(...) conceptual unification (...)"
[9] Frege's levels are denoted Sprache, Sinn, and Bedeutung, which in English philosophical terminology is language, meaning and reference.
[10] Begriffschrift, a language for pure concepts

1.2.2 Conceptual analysis after Frege

After Frege, conceptual analysis in philosophy, has gone in two directions with respect to the role formalization play in the analysis. The the first direction is theories of meaning[11] and reference[12], which we shall not consider in this paper, and the second is applied logic.[13] Frege's work relates more closely to mathematics than ordinary language and its semantics.

Applied logic grows out of Frege's work as the extension of formal methods applied to the domain of common concepts in ordinary language. Applied logic takes the concepts transcending first order logic serious, by introducing new operators capturing concepts like *obligation* and *permission* [42], *knowledge* and *beliefs* [17], *time* and *computation* [16], *typical situation*, *action* [36] and *counterfactual conditionals* [24].

First order logic, and even propositional logic has its anomalies as shown in the paradox of the augmentation-principle:

> If the postman comes then you will get your letter. Then by propositional logic we infer: If the postman comes and he burns the letter, then you will get your letter

This anomaly of the material implication can be solved by introducing a new modal conditional and a new set of axioms where strengthening the antecedent is rejected. But other anomalies may arise from the new system, where implication is given a domain specific meaning.

There are two observation related to this example. First, the example itself serves as an benchmark for testing theories, axiomatizations of non-monotone reasoning that philosophers might come up with. The set of benchmarks forms a finite set. Compared to classical theory of science, the benchmarks play the role of observations sentences, on which the theories (formalism) are calibrated. Second, the method of formalizing, testing, reformalizing, invent new benchmarks and then testing, reformalizing, testing, and so on, can be described as a circle movement. Progress in understanding is achieved by a back and forth movement, where the calculus change, but the set of benchmarks remains a relatively stable finite set.

1.2.3 Conceptual analysis as part of mature sciences

Conceptual analysis is not foreign to science before Frege. The treatment of infinitesimals in the differential calculus in the 17'th century, the scientific dispute preceding and including Lebnitz and Newton, was a great achievement in mathematics and physics, where the refinement of the mathematical concepts in the differential calculus where guarded by conceptual analysis. By doing this, they prepared the way for modern analytic geometry.

[11] The theories of what 'meaning' and truth is.

[12] The theories concerned with the question, how can it be that names denote to objects in the world.

[13] Some might claim that this is not the case, that there is no real sharp distinction between the two directions. We find it fruitful to make the distinction because of the difference in perspective, technical skills and methods of the two groups.

1.3 Reasons for doing Conceptual Analysis

Concepts are not given to us a priori, as precise and well understood. Concepts are constructed by humans.

Computer Science is a young science[14] There are at least four interrelated reasons for 'conceptual confusion' in computer science, which are *the immaturity of the science*, *the layers of abstraction*, *the quantity involved in computing*, *the rapid development of technology*.

First, computer science is a true hybrid of several sciences, physics, discrete mathematics and social sciences. To be more specific *electronics*, in searching for faster, smaller and more efficient hardware, *mathematical logic*, in designing circuits and designing programming languages, *cognitive psychology* and *sociology* to understand human interaction with the computer. The goal of research is to make computers work more efficiently in helping us solving practical problems in our daily life and at work.

Second, the use and design of computers relies on the notion of 'layer' of abstractions. To use in layers of abstractions means to use concrete and abstract concepts. But there seems to be too little awareness of how the creation of concepts take place and how concepts develop over time in computer science, as seen from the community itself.[15]

Third, and most importantly, new technology and concepts evolve in a true egalitarian way. Compared to the classical scientific disciplines, computer science is brought forwards by hackers, businessmen and managers and engineers, and to a smaller extent by theoretical computer scientists. Scientific progress in computer science can be described by the processes of both building concepts and technology. Although the scientific communities contributes to foundational research, there is an increasing tendency that the evolution of existing concepts , new concepts and technology (understood in the widest sense) is driven forwards by the mass of people in the computer business outside the classical research institutes and universities.

Fourth, the rapid development of technology itself, relies on the capability of the contributers of the development to conceptualize what they are doing. Understanding means conceptualizing. Conceptualizing means building new concepts.

The consequence of this as seen from the perspective of computing engineering and computer science is a landscape of too many and too unclear concept. It is rather the rule than the exception that a scientific term has more than one meaning, and hence unclear meaning, or that several names have the same meaning in different communities of engineers and computer scientists.

[14] If it can be classified as a science at all. Computer science today is characterized more in its plurality of methods and divergence of perspectives, than a clear understanding of method and a limited domain of discourse.

[15] Theoretical computer science is of course an exception, but we belong to a minority. Both from the perspective of education and industry theoretical computer scientists are pushed more and out in the dark. Evidence for this can be found throughout Europe, positions in theoretical computer science are withdrawn, and the companies apply for candidates with very technology-specific skills.

1.3.1 Being formal about informal matter

To do conceptual analysis in computer science (understood in the widest sense) means to be confronted with actual usage of concepts in both informal discussions and scientific work. More than being a logical, platonical investigation of the internal relationship of the concepts them self, we have to investigate empirically the actual usage of concepts an that the change of the informal semantics of concepts.

The overflow of concepts and the diversity of meanings imposes the need for working in another direction than the usual way, as done in software engineering. It indicates the need for limitation. One such limitation is the decision to stay inside a formal language. A formal language is precise. A sound formal language can serve a tool for calibrating our understanding of a concept. In general concepts are not precise and their usages normally diverge. In the research frontier of applied computer science this is rather the rule than the exception.[16]

1.3.2 Three perspectives on formalization

The outcome of a process of formalizing a domain can be threefold, *descriptive*, *weakly normative* and *strongly normative*.

Having a *descriptive* perspective means to apply a formalism in order to give a taxonomy of a domain, so that we can reason about the domain and prove facts about it. Two examples fit the descriptive view. Frege's investigation of the laws of thought is an uncovering of a platonic reality. Verification of programs, to use formal tool in order to discover critical or dangerous consequences of running a program that controls a nuclear power plant.[17]

To be *weakly normative*, means to investigate the usage of concepts and indicate, by finding inconsistency and incoherence in meaning, how usage could be changed in order to achieve clarity.

A *strongly normative* perspective means to investigate usage of a set of concepts and then using a formal language to specify a protocol, standard, programming language or a design paradigm, which everybody is supposed to follow. The normativity lies in the commitment for the users or programmers in applying the implemented version of the formal system.

RM-ODP is an standardization where the objective is "the developments of standards that allows the benefits of distribution of information processing services to be realized in an environment of heterogeneous IT resources and multiple organizational domains".[18] It not clear whether RM-ODP could be considered to be strongly normative, but the way they formulate their objectives and motivation strongly indicates so. In RM-ODP, the Foundations, Architecture and Architectural semantics are all intended to be normative[19], but normative in what sense? The conceptual framework "is based on precise concepts derived from current distributed processing developments and, as far as possible, on

[16] Mathematics is of course an exception, but not many people have time to listen to the mathematicians in the computing industry.

[17] One could of course claim that the verification showing a dangerous configuration itself is normative to the program itself, by committing the programmer to go and fix the bug in the program.

[18] [18] p. 6

[19] [18] p. ii

the use of formal description techniques for specification of the architecture".[20] The rapid development of new technology undermines the work of committee's like the Open Software Foundation[21] or the Object Management Group.[22] Although the ISO-OSI standard was intended to be strongly normative, nobody follows it, and it is not likely to believe that somebody will in the future. A common situation in computer science is that intended

1.3.3 Conceptual archaeology

Our scientific method will therefore be to use the elements in the formalism as spades and hoes to dig out the meaning of concepts in computer science. Where no coherent meaning can be found we shall give extrapolations and refinements. We shall call our method 'conceptual archaeology'.

- The tools will determine the objects: The difference in expressibility of the systems would give us different shades of the concepts or genuinely different concepts.

  This is similar to the archaeologist using spades to dig in order to uncover an ancient building, but missing an golden ring. If she had used a spoon she would have found the ring but missed the building.

- Start of digging: An investigation of concepts must begin somewhere. The expected layers of diverging meaning covering a concept must be unraveled, but we should be careful not being limited in our investigation by the first community of computer scientists we ask for the meaning of a concept.

  The archaeologist must seek for the most reasonable place and the appropriate tool to start digging. A good archaeologist has an intuition for the landscape, where to start, and when to restart the search in another place in order to find more interesting objects.

- Realism and humbleness: Rather than being strongly normative with respect to the Domain of Discourse, our intention is to clarify and suggest reasonable interpretations that are founded in formal systems of concepts not including the concept for investigation. That is, we pretend only to be weakly normative with respect to the outcome of our analysis.

  The attitude of an archaeologist when finding a historical object is characterized by humbleness and curiosity in the interpretation of the objects for investigation.

- Phases of reflection and action: The work will shift between pure conceptual analysis - reflection and the hard work trying to make the output of the conceptual analysis fit into an adequate formalism.

  An archaeologist is both a scientist, interpreting and reflecting on the objects she finds, and a practical worker not afraid of getting dirty and tired in searching.

[20] [18] p. ii
[21] OSF for short
[22] OMG for short

1.4 What 'session' might be

A short journey in the literature and on the web gives no clear understanding of the concept 'session'. It is a concept changing meaning gradually, covering new phenomena when the time goes on.

In the reference model for Open Distributed Processing [19] we do not find 'session'. But 'Liaison' has family resemblance with some of our interpretations of . 'Liaison' is defined by contractual context.

- Contractual context: the knowledge that a particular contract is in place, and thus that a particular behavior of a set of objects is required. An object may be in a number of contractual contexts simultaneously: the behavior is constrained to the intersection of the behaviors prescribed by each contractual context.[23] [24]

- Liaison: The relationship between a set of objects which results from the performance of some established behavior; the state of having a contractual context in common.[25]

But, this does not really help us very much. The problem is that RM-ODP, ideologically, is so closely related to the paradigm of object-orientation, so close that it might be an obstacle rather than an advantage to use the concepts in RM-ODP to explain or define 'session'. "Every ODP system specification is based on the concept of objects" [18] p. 11. But it does seem to be the case that 'session' should be understood entirely in the paradigm of object-orientation. It might although be the case that one could *interpret* the concept in the paradigm of object-orientation.

Second, we could look up in a dictionary and find its meaning: In Oxford Advanced Learner's Dictionary we read that 'session' among other things means "single continuous period spent in one activity". This captures the one of the common knowledges of how to use the word, and should be taken

Thirdly, one could look up in a canonical book on Computer networks and find that 'session' is a layer in a model for network-architecture that nobody uses anymore. In [38] p. 32-33 and [37] page 253-256 the concept session appears respectively as the 5'th layer in the ISO-OSI reference model and in the context of the discussion on the problems with file sharing in a distributed system. The important capabilities of the session layer is threefold, dialog control, synchronization and resynchronization. To have *dialog control* meant that, in the old network architecture where two agents communicating had to share the same channel, there had to be a control mechanism deciding which agent that where supposed to communicate (transmit) at the moment. To *synchronize* meant to set timestamps on the data to be transfered over the channel, in order to be

[23] [19] p. 13

[24] The examples of liaisons which result from different establishing behaviors are
a) a dialogue,
b) a binding,
c) a transaction,
d) an (N)-connection,
e) an association between (N)-entities enabling them to participate in (N) connection-less communication (as in OSI) f) a relationship between files and processes which access the files

[25] [19] p. 13

able to *resynchronize*, which meant that the session layer had capabilities of retransmitting lost packages from the sender to receiver, by using the timestamps known by both.

1.4.1 Examples of sessions

Instead of looking up in a book telling us what the concept really is, we could take the empirical approach and search after actual and potential usages of the concept. Let us describe some contexts of usage that is called sessions, or might be called sessions in the future:

- I have logged in on my computer without any network-connection.[26]
- I am logging in, then running several processes on a server machine from my computer at my office, and log off at the end of the day.[27]
- An ultra thin client, on which I can use a special card keeping my ID to access my session from the ultra thin client next door, and continue to work within the same context of programs running on the server.[28]
- I use special software Y to access a server-machine from a terminal X_1 equipped with an operating system Z_1, get an ID for this session and use it for a while, and then log off the and the next day I connect via a a terminal X_2, running an operating system Z_2, and run the software Y on X_2 by the same ID as yesterday and continue to work with the same processes running on the server machine.[29]
- I start writing a letter to a friend on my stationary computer, but have to leave in a hurry, so I continue to write on my portable computer equipped with a antenna keeping the connection to the server computer. After some minutes somebody must borrow my lap-top and therefore I continue to work on the letter on my little Personal Digital Assistant, still connected to my original server, being able to look what I have written up till now[30]
- I join a MUD-like game, that started at time t_0 on a server machine X, and start playing at time t_1, but get killed at time t_2 which is game over for me. The game itself will go on for some while and end at t_3. My participation in the game is certainly a session. But it is likely to suggest that the game or the administrator relation to the game is a session as well.[31]
- I join an ongoing MUD-game just like above, but while the game proceed, the Game-administrator moves the game-server from its original terminal to a new one.

[26] Personal session on a personal computer at home.

[27] A session on a remote time sharing system.

[28] Session mobility, realized in network of Sunray terminals connected to a Sunray-server, where a Sunray-card gives you access to your session.

[29] Sessions are kept alive even though one is logged off. VNC - Virtual Network Computing. An ID-number chosen by you, and the UNIX server machine memorizes your processes and when requested, unfolds the encapsulated processes again. My request the next day can be from a MacIntosh and I only need to download a VNC client for this specific machine to have a graphical user interface exactly the same as yesterday.

[30] Seamless movement of a session through several devices.

[31] Distributed sessions.

- I am a participant in a real-time game over the Internet, where an arbitrary number players interact in real-time where the whole game event appears as a continuous event for every participants, in spite of the delay on the network connecting the players.[32]

1.4.2 The primitive concepts

The examples given above might suggest a preliminary informal definition or rudimentary description of 'session':

> A session is an event limited in time. It is a temporal concept, with a start-point and an endpoint. A session is a relation between a subject and a possible many objects. A session involves concepts of communication.

To know the meaning of a concept, is to know what falls under the concept and that which does not. It entails the capability of making distinctions. One way to do this, is to find the primitive concepts that can be used to define the concept, but are theoretically simpler. A suggestion might be:

> time, place, state, communication, interpretation, identity, process

1.4.3 Five questions one could ask about the concept 'session'

- What is an example of the simplest session?
- Which other fundamental concepts would session rely on?
- Should it be possible to fork and merge sessions?
- Are there any equality predicates for sessions?
- Are sessions ordered in hierarchies?

The questions are both ontological and algebraic. They are ontological in the sense that the answers will determine which events fall under the concept and which does not, in other words they will determine what 'session' is. They are algebraic in the sense that confirmations to the questions will unfold algebraic properties of session-expressions as formulated in a formal theory.[33]

1.5 Concluding remarks

Our group is settled in an applied Research Institute where one of our concerns is inventing new ideas for applications and speculates about future requirements and technologies for mobile systems. There are two reasons why we consider Pi-calculus to be a promising framework to stay inside.[34]

[32] This is a very hard technological challenge. There are no really good solutions at the moment on how to do this.

[33] Where sentences in a language of sessions compile down to the language of π-calculus or some similar system.

[34] Nisse Husberg proposed to us that petri-nets could be a fruitful framework to stay inside, and we shall investigate this track in the future.

First, Pi-calculus is a formal calculus expressed in an abstract language, that gives the opportunity of stating technology independent properties of a mobile system or an application running on a mobile device. Secondly, technology evolves rapidly and freely with a continuous conflict between the need for standardization and specialization, between compatibility and incompatibility involving competing hardware, software and telecom-companies.

We suspect that formal methods has a role to play in on-the-edge technology that we are exposed to where the needs for tools for thinking is critical. As discussed above, we also expect to find several 'session'- concepts, and hopefully a unifying framework to reason about a subset of these. But time will show whether we succeed or not.

Chapter 2

Mobility framework

2.1 Introduction

Advances in wireless technology and mobile information appliances and the emergence of the agent paradigm have added a new dimension to distributed computing, namely, mobility.

Work in mobility comes in two flavors: one concerned with mobile code and the other with mobile appliances. Mobile code, most commonly referred to in the literature as (mobile) agent, is a piece of code that can (autonomously) move around the network from node to node while executing. In this report, we use the same terminology as the one in [4] to refer to these two forms of mobility, i.e., *mobile computation* and *mobile computing* respectively.

Mobile computation deals with issues involved in agents moving between hosts and their communication, both with the user and among themselves, and security issues from the point of view of both the agents and the host environments.

Mobile computing is concerned with computation carried out on mobile devices in the light of the issues arising from limited resources, low bandwidth and the intermittence of the connections of such devices to the fixed network. Security issues are, of course, of great importance here as well.

Work in mobile computation has led to the specification of two standards: the OMG's[1] Mobile Agent System Interoperability Facility (MASIF) and the FIPA's[2] series of standards on agent communication, management and message transport.

Originally, FIPA's standards dealt with issues related more to intelligent agents while OMG's standard dealt with mobile agents. But FIPA has in the last years addressed the issue of mobility with MASIF in mind.

Research in mobile computing has resulted in many prototypes mainly based on the client-server paradigm and addressing different aspects of mobile computing. A taxonomy of the work in that area can be found in [20]. But much of the work on mobility has tackled the one or the other aspect of it and very little has been done in combining both.

This chapter aims at defining a suitable framework for supporting mobile

[1] OMG stands for Object Management Group
[2] FIPA stands for Foundation of Intelligent Physical Agents

users, *mobility framework*, by identifying the concepts essential to such a framework.

2.2 A Framework for Supporting Mobile Users

In a mobile setting, in general, an end-user using a portable appliance accesses data on or uses a service from a server located at a host in the fixed network. The client-server paradigm is therefore adequate for systems supporting mobile users; but it has to be tuned to the peculiarities of the mobile environment.

The premises of mobile computing differ from those of the conventional distributed computing in that

- the connectivity is weak, i.e., the connection to the host is intermittent with fluctuating bandwidth that can become quite scarce,
- most often, mobile hosts dispose of limited resources, and
- the users are nomadic.

The first two items in the list above have to do with technological constraints while the last one is concerned with mobile users' work patterns and the requirements they give rise to.

The mobility of the users, the intermittence of the connections and the variations in the available bandwidth make the mobile environment quite dynamic. Fixed location and connection cannot be assumed anymore. The fact that the clients and servers can physically be located very far apart together with the possibly narrow bandwidth available make the network latency non-negligible.

To deal with the dynamism in the environment and the latency in the network are the main challenges of mobile applications. Therefore there is a need for a framework that provides support to mobile users.

We shall now take a closer look at the characteristics of a mobile environment and study their consequences for a mobility framework.

2.3 Weak Connectivity

On the one hand, the limited resources of mobile devices makes mobile applications very dependent on access to remote servers in the fixed network. For example, a mobile user on a trip might suddenly need data not available locally on his device. On the other hand, the necessity to cope with weak connectivity points toward making them self-sufficient, i.e., capable of continued operation while disconnected from the network.

In other words, applications operating in a weakly connected environment, should be resilient to connectivity failures. That is, clients should be able to continue their operation even in the absence of a server, e.g., after a connection failure, in a disconnected mode.

There exists a vast body of work concerned with weak connectivity. A brief discussion of work representing the main different approaches to the problem is presented below.

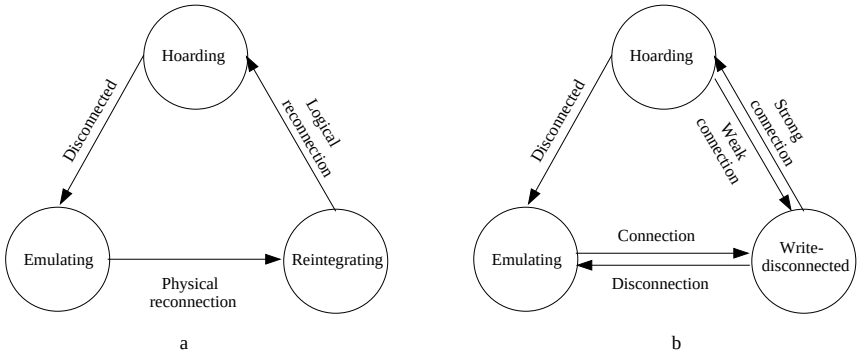


Figure 2.1: State Transitions in the Coda System.

2.3.1 The Coda File System

Coda [35, 3] is a distributed file system that provides clients with continued access to data even in the event of server or network failures. It introduces the notion of *disconnected operation.*

Coda operates in one of the states [34]: *hoarding*, *emulating* and *reintegrating.* It is usually in the hoarding state ready to cache critical data in the event of connection failure. In the case of disconnection, it enters the emulating state. In this state the client-side of the Coda system emulates its server-side making possible continued operation. Changes made by the user to the file system in this state should be made persistent and are logged to this end. Upon reconnection, Coda enters the reintegrating state submitting to the server the changes made during disconnection and then enters the hoarding state again.

Note that in the above discussion, the reintegration state is a transient state and the system will eventually proceed to the hoarding state. In the latest version of Coda, reintegration is done by a background process that propagates updates to the server asynchronously. It has therefore become a stable state called *write-disconnected* and the transitions among different states have changed. The state transitions in the two versions of Coda are depicted in Figure 2.1.

The Coda server might therefore receive conflicting updates to the file systems from different clients. Coda's earlier version only incorporated some automated conflict resolution mechanism at the directory level. Conflicts that could not be resolved were handled by human intervention. Later, *application-specific resolvers* (ASRs) were adopted. This mechanism was first introduced by the Bayou system (presented next). The ASRs allow applications to leverage their semantic knowledge and provide conflict-resolvers, programs, that are installed and transparently invoked when necessary.

2.3.2 The Bayou System

Bayou [8] is a replicated, weakly consistent storage system in a mobile environment. One of its main design goals is to support non-real-time collaborative applications for disconnected work groups. In Bayou not only clients but servers are mobile as well.

It is based on a read-any/write-any style of access. That is, in the Bayou

system each data collection is replicated at a number of servers. Bayou clients can read data from any server and write (insert, modify and delete) data to any server – not necessarily the server they read from. Once a write is accepted by a server, the client has no more responsibility for that write.

To achieve its goal, Bayou provides mechanisms that enable application specific detection and resolution of update conflicts [39], i.e., *dependency checks* and *merge procedures* respectively, and a protocol by which the resolution of update conflicts stabilizes thus ensuring the consistency of the replicated data collections.

It introduces the notions of

tentative data [41]: a write accepted by a server from a client is initially considered as tentative. Eventually, it will be *committed* when the server can observe that it has received and applied any write that precedes the one at hand. This process of committing writes is made possible by the protocol mentioned above. Users are made aware of the state of the data, i.e., whether it is tentative.

session guarantees [40]: weakly consistent replicated storage systems, especially with a read-any/write-any style of access, have the disadvantages of confusing the users. For example, a user may read a data value and then later read an older value for the same data (from another server). Session guarantees aim at alleviating this problem by presenting an individual application with a view of the database that is consistent with its own operations during a *session*. A session is an abstraction of a sequence of read and write operations performed by an application during its execution. The following four guarantees are supported:

- *Read Your Writes* – read operations reflect previous writes.
- *Monotonic Reads* – successive reads reflect a non-decreasing set of writes.
- *Writes Follow Reads* – writes are propagated after reads on which they depend.
- *Monotonic Writes* – writes are propagated after writes that logically precede them.

2.3.3 The Rover Toolkit

The Rover toolkit [21, 22, 23] is a general toolkit for building mobile applications. It does so by providing two basic mechanisms: *relocatable dynamic objects* (RDOs) and *queued remote procedure call* (QRPC).

The RDOs are objects with a well-defined interface and they can dynamically be loaded from servers to clients or vice versa. To operate in a disconnected mode, a client application can load all needed RDOs from the server. Changes made to them are propagated back to the server upon reconnection using QRPCs. Rover also provides support for application-specific resolution. The Rover toolkit is described in more detail in Section 2.4.2.

2.3.4 Support for Weak Connectivity

Mobile clients often operate in a weakly connected networking environment. To ensure their continued operation under these circumstances has been the goal of many systems, some of them discussed above.

In general, the elements involved in the solutions presented by those systems are the same in essence although different in realization. The main ones are:

- caching remote data along with a mechanism for propagating the changes made to the cached data back to the server,
- mechanisms supporting detection and resolution of update conflicts, and
- emulation of the server at the client.

A mobility framework should therefore incorporate support for the elements listed above in a general way. That is, it should not dictate particular strategies but provide general mechanisms.

There is no one caching strategy that suits all types of applications. Therefore, each application must decide on its own caching strategy. On the other hand, the framework should support cache management, logging of changes made to the cache, propagation of those changes to the appropriate server(s), application-specific conflict detection and resolution, and installation of server emulations, i.e., relocation of functionality.

2.4 Limited Resources

Most of the wireless mobile hosts have limited local resources as well as access to narrow bandwidth only. In addition, the availability of these resources may vary over time, e.g., when running a number of services in parallel or when experiencing fluctuations in the bandwidth. Therefore, applications must be able to adapt themselves to the changes in the environment.

An extreme case of limited resources is in fact the absence of a resource altogether. For example, in the absence of bandwidth, a mobile client should operate in a disconnected mode, discussed earlier, or when the computational power is very limited, all computation should be done at the server.

The range of adaptation strategies, shown in Figure 2.4, varies widely among existing systems. It can range from client-only adaptation to server-only adaptation passing via different degrees of cooperation between the client and the server supported by the mobility framework. This last category is called *mobile-aware*.

In the client-only adaptation strategy, called also *laissez-faire* [31], clients alone are responsible for the handling of the mobile environment. Such clients in general are not concerned about coexistence with other clients and therefore compete with each other in a destructive way. For example, all clients running on a mobile device compete for the same resources; in the absence of some form of cooperation among the clients, each of them will try to hold as much resources as it needs, reducing the overall performance.

In the server-only adaptation strategy, the server hides the mobile environment from the client, taking all decisions itself. In this way, existing clients can be used on mobile devices without any (radical) changes. This is achieved by

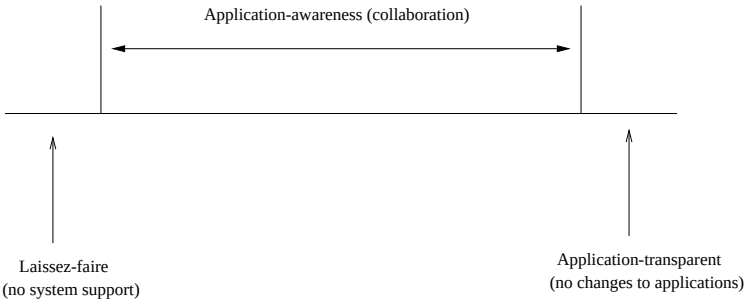


Figure 2.2: The spectrum of adaptation strategies.

placing a proxy server, between the client and the server. The proxy presents to the client the same interface as the original server and processes the client requests by accessing the server and handling all issues related to the mobile environment. The proxy server has usually a client side and a server side.

The former strategy hinders a fair distribution of resources among concurrent client applications, on the same device, while the latter strategy is not flexible enough and prevents the clients, which know best about their own states and available resources, from participating in the decision making. The best strategy, in most cases, is therefore collaboration between the client and the server with support from the framework for a more efficient resource management.

Different approaches have been taken to tackle the adaptation problem. We shall now try to establish the main characteristics of a mobile-aware system by studying a couple of existing examples.

2.4.1 The Odyssey System

The Odyssey system [31, 30], designed to support mobile information access, focuses on adaptation with respect to the quality of data. The adaptive decisions lie at the clients and are concerned with the quality of data received while the servers are able to provide data in accordance with the chosen quality. For example, as a result of a drop in the available bandwidth, a client might decide to degrade the quality of the received video stream by accepting a reduced size for each individual frame. The clients cooperate with the Odyssey platform to choose a suitable quality, i.e., the clients have an adaptation policy and based on it, they inform Odyssey of their resource requirements; Odyssey is in charge of resource allocation, notifying the clients of resource changes, and enforcing the adaptation policies.

The Odyssey system does not really tackle the disconnected mode. Of course, an Odyssey client can, if enough memory is available, cache in all the data it needs in advance and then proceed in a disconnected mode but this is not its main goal.

2.4.2 The Rover Toolkit

The Rover toolkit presents the applications with a distributed object system. A Rover client application runs usually on a mobile host (it can be run on a stationary host as well) while a Rover server application typically runs on a

stationary host. To deal with limited resources, it provides *relocatable dynamic objects* (RDOs) and *queued remote procedure call* (QRPC).

The RDOs are objects with a well-defined interface (they can be either very simple or complex) that can dynamically be loaded from servers to clients or vice versa. Their purpose is manifold. For example, clients can load RDOs from servers to reduce communication or load RDOs into the servers to carry out heavy computation or complex actions against data at the server.

Updates made to the RDOs are propagated to the Rover server using the QRPC mechanism. The Rover server has the main, *canonical*, copy of the RDOs and applies the received QRPCs to them.

Loading RDOs representing the necessary server functionality into the client from the server, allows the client to operate in disconnected mode. Updates to the cached RDOs are, as mentioned above, remembered by the QRPC mechanism and propagated to the server upon reconnection.

The Rover toolkit provides general mechanisms and thus supports implementation of the whole spectrum of mobile-transparent to mobile-aware applications.

2.4.3 Support for Limited Resources

As said earlier, clients know best about their available resources and how to manage them. Some clients do not only have limited resources, but also limited capabilities. That is, they cannot handle all types or format of data. For example, a regular mobile telephone cannot handle video, or a device might only be able to display pictures stored in a particular data format (e.g., gif, jpg, etc.).

A framework for mobile computation should therefore alleviate the problem of limited resources at the clients by allowing them to tailor their activities accordingly. It should enable the client to

- state its capabilities,
- state the quality of the data it can currently handle,
- delegate computation to the server (generally with considerably more resources) if and when necessary.

The first two items above deal with bandwidth scarcity while the third deals with both bandwidth scarcity, limited memory and computational resources. It is important for the client to inform the server about its limitations so that inappropriate data is not sent and bandwidth is not wasted.

2.5 Nomadic Users

The two previous sections, Sections 2.3 and 2.4, described how a mobility framework should deal with the technological constraints imposed by mobile environments. This section is concerned with how to best support users in carrying out their tasks efficiently in such environments.

Nomadic users can be characterized by

- being away from their home base, i.e., their regular place of work (they might even not have one),
- using mobile equipment,
- being disconnected for long periods of time due, e.g., to unavailability of the network, their devices being out of power, etc., and
- possibly changing equipment.

A major consequence of mobile environments for the nomadic users is reduced data availability. Therefore, support for efficient ways of obtaining the desired data is of importance. Nomadic users are often in situations where it is desirable to carry out various tasks at the server machine either because of lacking computational power of the mobile device, necessity to bring computation close to the involved data, the desire to perform many tasks in parallel, or simply not being able to use the mobile device for a period of time. A typical scenario would be that a nomadic user launches some task to be carried out at the server while he is busy with some other task. Later in the day, he will pick up the result.

Another aspect of nomadic work is that nomadic users might be using different mobile equipment at different times. Ideally, they should be able to go to another device and continue the work they had started at the previous device. This seems not to be possible with the existing systems.

In a mobility framework, the key in supporting nomadic users is thus flexibility. That is, nomadic users should be provided with mechanisms such that they can tailor their work pattern to the situation at hand.

2.5.1 Support for Nomadic Users

Two concepts that seem to provide the desired level of flexibility to nomadic users are:

- mobile agents, and
- a concept of *session* allowing them to continue the same task at a new device.

Mobile agents can operate independently of and also in parallel with their originators. Using mobile agents, nomadic users have thus the possibility of *duplicating* themselves to carry out multiple tasks in parallel, to carry out tasks while they are not available themselves and to carry out the computation close to the data or at servers with more computational resources.

A client should therefore be able to send a mobile agent to a server and receive the agent back with the desired result. The server should perform security checks on the agent upon its arrival and let it perform its task with respect to the outcome of the security checks, and send it to its next destination, most probably its originator, when requested to do so.

Most of the work done on mobile computing is concerned with accessing shared data. Only the Rover toolkit touches on the aspect of transferring computation between clients and servers. The RDOs of the Rover toolkit provide support for some of the situations listed above.

There are many platforms supporting mobile agents, but they are not concerned with mobile environments and thus lack support for the other issues, discussed earlier, that are related to mobile environments.

As stated above, the other dimension in providing flexibility is to support nomadic users in continuing their tasks on a new device. The concept of session for supporting this is elaborated in the next section.

2.6 Session

A session is a concept orthogonal to mobile computing and can as well be used in conventional environments.

The concept of session is not focused much, at least on a logical level, in conventional environments mainly because of the availability of resources and steadiness in connections.

Likewise, in the literature on mobile computing, there is very little emphasis on the concept of session. All servers keep some kind of information about the different clients they serve and all clients keep information about their own states, but the notion of session is treated rather implicitly. Some of the systems explicitly define sessions for very specific purposes. The Bayou system, e.g., defines a session as [40]

> ... an abstraction for the sequence of read or write operations performed during the execution of an application. Sessions are not intended to correspond to atomic transactions that ensure atomicity and serializability. Instead, the intent is to present individual applications with a view of the database that is consistent with their own actions, ...

This definition is in order to define and provide session guarantees as described in Section 2.3.2.

The Rover toolkit also has a notion of session which is based on Bayou's session. Its API contains a *create session* method which returns a session identifier. The notion of session is used to specify consistency requirements prevailing throughout the session.

None of the systems mentioned in this report supports change of device for continuing the same work, though this would offer great flexibility to nomadic users.

To support the flexibility of changing devices, the notion of session should be treated as a general high-level concept. It should be defined such that it also supports the existing definitions such as that of the Bayou system.

It is hard to define a session in terms of when it starts and when it ends, in a general way suitable to all applications or services. Some services might have a semantic that lends itself to a natural and logical definition of the start and end of a session, but most services do not.

For example, when sending a mobile agent off, it seems logical to assume that a session is from the time the agent is launched to the time it has returned and delivered the result to the user. But when working on a shared document, there is no straightforward way to define a session. We feel that the user best knows the context and the semantics of the task at hand and therefore is best positioned to define the start and end of the session. Ideally, a user should be

able to run many sessions in parallel, some of them possibly involving the same application.

We define the session as follows:

> A session is as a sequence of active periods (in terms of the client performing tasks) separated by inactive periods, which can be very long in a mobile setting. The active periods can take place in either of connected or disconnected modes. Moreover, as stated by the Bayou project, sessions are not intended to correspond to atomic transactions that ensure atomicity and serializability.

We shall now take a closer look at the elements of a session and how it can be supported in a mobile environment.

2.6.1 Elements of a Session

According to the given definition, a session is comprised of a sequence of active periods. A session needs therefore a memory to recall where it left off in its previous active period. In other words, a session has a *state*.

Some of the ingredients of this state are common to all sessions while others depend on the session's application. In the following a more detailed description of these two parts of a session state is given and the impact of session on the framework is described.

Application Independent Elements of the State

To continue work on a previously started session, one needs to identify that session. That is, a session must have a unique identifier (session ID or SID) that lasts throughout its life. The session ID is issued by the server when a session is started. The server holds the states of all the sessions it participates in and manages them.

A server should know about the capabilities of the client device it is servicing, in terms of the data types or formats that it can or cannot handle. It can use this information to save bandwidth by sending only the relevant type of data in the appropriate format to the client.

It also should know who the end-user is. This information can be, e.g., a user-name/password combination. The server can use this information in various ways, such as for billing or for security checks.

To access information on the server that might not be public, the user needs to present the server with his credentials describing the access rights and privileges he has. This information can be tied to the user-name/password of an user or it can be some separate piece of information.

An identification of the service (application) offered in a session is also part of the session state. In addition, the server needs an identification of the client's device to be able to notice a change of equipment and the address at which the client can be accessed (e.g., IP address, phone number, etc.). This address can be used to establish contact with the client if necessary or to put others looking for the corresponding user in touch with him, if such a service is desirable.

Application-Dependent Elements of the State

The application-dependent elements of the session state can be defined in terms of one general requirement. They should hold enough information to reinstate the state of the session to what it was at the end of the previous active session on any machine at any location. For example, if the session ran a distributed file service, the state should contain a list of all files cached at the client. In the case of the user continuing his work on a new device, the server will use that information to load all the files into the new device.

The ability to operate in a disconnected mode entails additional complexity for the transfer of a session from one device to another. If the last active period of the session was carried out in disconnected mode, then the server is unaware of actions undertaken in that period and its session state does not reflect them. The same is true if the previous active period has been terminated by, e.g., a sudden network failure. In most cases, some of the client actions are not committed to the server yet. The best the server can do is to reinstate the session state to the last active, non-disconnected period.

Clients keep record of non-committed operations and commit them upon reconnection and thus update the state of the server. Therefore, continuing the work from the same client device does not cause any loss of information since the information missing at the server is at the client which brings the server up-to-date. This suggests that the application-dependent part of a session state has both a client-side and a server-side.

By continuing the work at another device, the client-side's state information, e.g., the set of non-committed operations, is not available at the new device anymore. That is, by changing device, the user might lose some of his work.

2.6.2 What Session Means for the Framework

Section 2.6.1 suggested the use of a unique server-generated session identifier (SID). The uniqueness of a SID is, of course, with respect to the server generating it. This section will elaborate on the pros and cons of the SID.

If a client is restricted to only one session with a given server, then a SID is no longer needed. Upon connecting to the server, the user identifies himself, the server uses the identification information (e.g., user-name/password) to find the corresponding session, notices that the physical address has changed, and therefore reinstates the client's state from its own session state. Such a restriction though highly reduces the flexibility of work for nomadic users and is thus not desirable.

If the client is allowed to establish more than one session with a server, in the absence of a SID, the above scenario still holds. The major difference is that the new client is reinstated with the states of all the sessions that the user had at that server. The user is not able to choose a particular session if he wishes so. In an environment where resources are scarce, this will lead to waste of valuable resources, such as bandwidth and memory.

Using a SID gives the user the flexibility to both establish many sessions and to be able to pick only one for further work at a new device. But the SID has the disadvantage that it should be transferred somehow from one device to another.

The first candidate that comes to mind for the transfer is the user. The SID

can be saved on, e.g., a CD, a floppy disk or in the user's memory, and carried to the new device.

A better and more robust alternative is to offer to the user a registry service where each server registers each new session, i.e., the SID along with necessary information allowing users to get hold of the needed SID.

The scheme discussed above does not solve the problem of loss of work due to disconnected mode of operation or a sudden network failure. The client can provide the user with the ability to checkpoint the client's state when he wants at a specified location, e.g., a CD, a floppy disk, etc, and to re-establish the client's state from the checkpoint. But the checkpointed state should be carried to the new equipment by the user. Of course, it is possible to have the checkpointed state saved at some kind of a server as well, but this alternative might not be very efficient. Some hands-on experience with implementing a mobile framework is needed in order to test the alternatives.

According to the above discussion, the information that should be provided by a client upon establishment of a session with a server consists of the following parts.

- identification of the user (user-name/password),
- client device's address,
- client device's identification,
- client device's capabilities, and
- the desired service (application).

The client receives in return a server-generated SID, which it saves. On the server side, the server establishes a new session comprised of the generated SID, the information provided by the corresponding client, and the application dependent information.

2.7 Conclusion

The previous sections dealt with the impact of different aspects of mobile environments on a mobility framework and the necessary elements to support them. This section gives a summary of the desired characteristics and facilities of such a framework.

The major desired characteristic is flexibility both in terms of

- supporting a range of strategies and allowing the applications to decide on the appropriate ones, and
- allowing the nomadic user to adapt his work pattern to the situation at hand.

In order to achieve the first bullet in the list above, a mobility framework should not try to hide the characteristics of the mobile environment from the clients but provide them with mechanisms that enable them to tackle the dynamism and the shortcomings of the mobile environment as they see fit. For

example, it should allow to build mobile-aware as well as mobile-transparent services.

To satisfy the second bullet above, two elements were identified: an agent-like facility and the concept of session as presented in Section 2.6.

To summarize, a mobility platform should include support for

- disconnected mode of operation, which involves
  - support for caching remote data in terms of a cache management facility caching the necessary file, logging the changes made to them and propagating the changes to the server,
  - mechanisms for detection and resolution of update conflicts, and
  - emulation of the server at the client;
- limited resources, including
  - client adaptation to the changes in the mobile environment by notifying the client of the changes in the resources, negotiating with it resource usage, and enforcing the result of negotiations;
  - moving computation between clients and servers;
- agent facility;
- session as defined earlier;

In addition, support for security, which is not handled in this report, is necessary.

Chapter 3

The π-calculus

Our presentation of the π-calculus attempts a (partial) reconstruction of §9 in Milner's textbook [25], but we give a more rigorous treatment. Also, we limit ourselves to the π-calculus proper, while Milner develops the theory starting from automata theory.

We have presented the theory in a level of explicitness that would satisfy the reader being introduced to π-calculus for the first time. In concrete this means that our recursive definitions and inductive proofs should leave no doubt about the mathematical precision of the abstract objects in discussion.

3.1 Basic calculus

This section details with the π-calculus itself and its basic properties.

3.1.1 Syntax and substitution

Here we define the syntax of the π-calculus, that is, the structure of process expressions. We also define substitution on process expressions and state some properties of substitution that are needed later on.

Syntactic conventions: π is an action prefix; a, b, x, y, z are names; P, Q, R are process expressions; σ, ρ are substitutions. In addition we use subscripts or primes on the above. Like Milner we use these symbols as both meta-variables and in the concrete syntax; distinguishing the two leads to a lot of extra book-keeping without much gain. Meta-equality is used to state that two expressions are syntactically identical, $P = P'$.

We have one piece of non-standard meta-notation: we sometimes use a 'underline' to denote which part of a process expression that will change when we reason about process expressions, for example:

$$\underline{\mathsf{0}|}a.\mathsf{0} \equiv a.\mathsf{0}$$

Note that the underlined part need not be a process expression, in fact, here it is not.

Definition 3.1 *There is an infinite set of names $\mathcal{N} = \{x, y, z, \dots\}$. The action prefixes are $\pi = \{x(y)|x, y \in \mathcal{N}\} \cup \{\bar{x}\langle y\rangle|x, y \in \mathcal{N}\} \cup \{\tau\}$.*

τ is called the unobservable action. Note that two occurrences of the same name may refer to two distinct variables.

Note that the symbol π is overloaded: it denotes the set of action prefixes, but is also used as a meta-variable standing for an action prefix, and finally it is used to stand for a proof-tree. The risk of confusion is negligible.

Definition 3.2 ($\mathcal{P}^\pi$) *The set $\mathcal{P}^\pi$ of process expressions is the smallest set such that*

$$\mathsf{0} \in \mathcal{P}^\pi \tag{3.1}$$

$$\textit{If } P \in \mathcal{P}^\pi \textit{ and } a \in \mathcal{N} \textit{ then } \mathsf{new}\, a\, P, !P \in \mathcal{P}^\pi \tag{3.2}$$

$$\textit{If } P_1, P_2 \in \mathcal{P}^\pi \textit{ then } P_1|P_2 \in \mathcal{P}^\pi \tag{3.3}$$

$$\textit{If, for every } i \in I(|I| = n), \textit{ it holds that } \pi_i \in \pi \textit{ and } P_i \in \mathcal{P}^\pi \textit{ then } \sum_{i \in I} \pi_i.P_i \in \mathcal{P}^\pi \tag{3.4}$$

We assume that a name a introduced by $\mathsf{new}\, a\, P$ is always globally unique.

When convenient we shall use $+$ since, $\sum_{i\in\{1,2\}} \pi_i.P_i = \pi_1.P_1 + \pi_2.P_2$. Hence;

Definition 3.3 *The process operators $\mathcal{PO}^\pi$ is the set $\mathcal{PO}^\pi = \{|, ., \mathsf{new}, +, !\}$.*

Definition 3.4 *The degree of process-expressions are defined by*

$$deg(\mathsf{0}) = 0 \tag{3.5}$$

$$deg(\pi.P) = deg(\mathsf{new}\, a\, P) = deg(!P) = deg(P) + 1 \tag{3.6}$$

$$deg(P_1|P_2) = deg(P_1 + P_2) = max(deg(P_1), deg(P_2)) + 1 \tag{3.7}$$

It is recursive to measure the number of occurrences of an operator in a process expression.

Definition 3.5 *Let $\odot \in \mathcal{PO}^\pi$ then $\mathrm{num}(\odot, P)$, where $P_1, P_2 \in \mathcal{P}^\pi$ is;*

$$\mathrm{num}(\odot, \mathsf{0}) = 0 \tag{3.8}$$

$$\mathrm{num}(\odot, \odot P_1) = \mathrm{num}(\odot, P_1) + 1 \tag{3.9}$$

$$\mathrm{num}(\odot, P_1 \odot P_2) = max(\mathrm{num}(\odot, P_1), \mathrm{num}(\odot, P_2)) + 1 \tag{3.10}$$

$$\textit{If } \sharp \neq \odot \textit{ then both } \mathrm{num}(\odot, \sharp P_1) = \mathrm{num}(\odot, P_1) \textit{ and } \mathrm{num}(\odot, P_1 \sharp P_2) = max(\mathrm{num}(\odot, P_1), \mathrm{num}(\odot, P_2)) \tag{3.11}$$

Syntactic conventions:

$$\pi = \pi.0$$

$$\mathsf{new}\, a_1 \cdots a_n\, P = \mathsf{new}(a_1 \cdots a_n)\, P$$

$$\mathsf{new}\, \vec{a}\, P = P \text{ or } \mathsf{new}\, a_1 \cdots a_n\, P$$

Notation $\mathsf{new}\, \vec{a}\, P$ represents a restriction over zero or more variables $\vec{a}$, where a restriction over zero variables means no restriction.

The π-calculus operators have precedence from high (tight binding) to low (loose binding) as follows:

$$. \quad ! \quad + \quad |$$

Operator new always takes a single process argument, using brackets if necessary. The following convention follows from the precedences above:

$$!\pi.P = !(\pi.P)$$

Definition 3.6 (Free names, fn) *The free names of process expressions:*

$$\mathrm{fn}(0) = \emptyset \tag{3.12}$$
$$\mathrm{fn}(x(y).P) = \{x\} \cup (\mathrm{fn}(P) \setminus \{y\}) \tag{3.13}$$
$$\mathrm{fn}(\bar{x}\langle y\rangle.P) = \{x, y\} \cup \mathrm{fn}(P) \tag{3.14}$$
$$\mathrm{fn}(\mathsf{new}\, a\, P) = \mathrm{fn}(P) \setminus \{a\} \tag{3.15}$$
$$\mathrm{fn}(!P) = \mathrm{fn}(P) \tag{3.16}$$
$$\mathrm{fn}(P_1|P_2) = \mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \tag{3.17}$$
$$\mathrm{fn}(\sum_{i\in I} \pi_i.P_i) = \bigcup_{i\in I} \mathrm{fn}(\pi_i.P_i) \tag{3.18}$$

The brackets are significant in Equation 3.13 above; we must ensure that, when $x = y$, it holds that $x \in \mathrm{fn}(x(x))$, that is, that the two occurrences of name x are identified as distinct variables.

Later we need a form of substitution on process expressions to define reaction between processes, and we also use properties of substitution in proofs. Substitution is known to be hard to get right and we therefore analyze it further.

The kind of substitution we use here is simpler than substitution in λ-calculus [33, §2]:

- we only substitute *variables for variables*, not general expressions for variables;
- substitutions may replace at most one variable (but several occurrences of that variable) in a process expression.

If $a, b \in \mathcal{N}$ then $\{b/a\}$ denotes a substitution. We use Milner's prefix notation for application. On names substitution has the obvious meaning:

$$\{b/a\}a = b \tag{3.19}$$
$$\{b/a\}x = x \qquad \text{when } x \neq a \tag{3.20}$$

The substitution $\{b/a\}$ replaces free occurrences of a with b in a process expression P. The following definition makes this precise.

Definition 3.7 (Substitution on process expressions)

$$\sigma \mathsf{0} = \mathsf{0} \tag{3.21}$$
$$\{b/a\}(x(a).P) = (\{b/a\}x)(a).P \tag{3.22}$$
$$\{b/a\}(x(y).P) = (\{b/a\}x)(y).\{b/a\}P \quad \textit{when } y \neq a, b \neq y \tag{3.23}$$
$$\{b/a\}(x(b).P) = (\{b/a\}x)(b').\{b/a\}\{b'/b\}P \quad \textit{where } y \neq a, b' \notin \mathrm{fn}(P) \tag{3.24}$$
$$\sigma(\bar{x}\langle y\rangle.P) = \overline{\sigma x}\langle \sigma y\rangle.\sigma P \tag{3.25}$$
$$\{b/a\}\,\mathsf{new}\,a\,P = \mathsf{new}\,a\,P \tag{3.26}$$
$$\{b/a\}\,\mathsf{new}\,x\,P = \mathsf{new}\,x\,\{b/a\}P \quad \textit{when } x \neq a, b \neq x \tag{3.27}$$
$$\{b/a\}\,\mathsf{new}\,b\,P = \mathsf{new}\,b'\,\{b/a\}\{b'/b\}P \quad \textit{when } x \neq a, b' \notin \mathrm{fn}(P) \tag{3.28}$$
$$\sigma!P = !\sigma P \tag{3.29}$$
$$\sigma(P_1|P_2) = \sigma P_1|\sigma P_2 \tag{3.30}$$
$$\sigma(\sum_{i\in I}\pi_i.P_i) = \sum_{i\in I}\sigma\pi_i.P_i \tag{3.31}$$

Syntactic conventions:

$$\sigma_1\sigma_2 P = \sigma_1(\sigma_2 P)$$
$$\{\}P = P \qquad \text{(empty substitution)}$$

Proposition 3.8 *Laws for substitution:*

$$\{a/a\} = \{\}$$
$$\sigma\{\} = \sigma$$
$$\{\}\sigma = \sigma$$
$$\{b/a\}\{a/b\} = \{b/a\}$$
$$\{x/b\}\{a/b\} = \{a/b\}$$
$$\{x/y\}\{a/b\} = \{(\{x/y\}a)/b\}\{x/y\} \quad \textit{when } b \neq y \textit{ and } b \neq x$$
$$\{a/b\}P = P \quad \textit{when } b \notin \mathrm{fn}(P)$$

The laws involving two substitution on the left-hand side are special cases of laws given by Mitchell [28, p. 54].

Proof The proofs are obvious; they are left as an exercise for the interested reader. □

3.1.2 Structural congruence

Definition 3.9 *If $M \in \mathcal{P}^\pi$, $a \in \mathcal{N}$ and $\pi_1 \in \pi$ then the set of elementary contexts $\mathcal{E}$ is the smallest set such that; $\pi_1.[\,] + M$, $M + \pi_1.[\,]$, $[\,]|M$, $M|[\,]$, $\mathsf{new}\,a\,[\,]$, $![\,] \in \mathcal{E}$.*

Definition 3.10 *If $M \in \mathcal{P}^\pi$, $a \in \mathcal{N}$ and $\pi_1 \in \pi$, then the set of π-process contexts $\mathcal{PC}^\pi$ is the smallest set such that*

$$\mathcal{E} \subset \mathcal{PC}^\pi \tag{3.32}$$
$$\textit{If } \mathcal{K} \in \mathcal{PC}^\pi \textit{ then } \pi_1.\mathcal{K} + M,\ M + \pi_1.\mathcal{K},\ \mathcal{K}|M,\ M|\mathcal{K},\ \mathsf{new}\,a\,\mathcal{K},\ !\mathcal{K} \in \mathcal{PC}^\pi \tag{3.33}$$

Notice that the set $\mathcal{PC}^\pi$ is a language of syntactic constructs. Note also that $\mathcal{PC}^\pi \cap \mathcal{P}^\pi$ is empty because there is always one and only one occurrence of the minimal context $[\,]$ in every $\mathcal{K} \in \mathcal{PC}^\pi$ (as easily shown through an inductive proof).

Definition 3.11 *The syntactic complexity* $\deg(\mathcal{K})$*, where* $\mathcal{K} \in \mathcal{PC}^\pi$ *is a mapping* $\deg(.) : \mathcal{PC}^\pi \to \mathbb{N}$*, defined recursively as:*

$$\deg([\,]) = 0 \tag{3.34}$$

$$\begin{aligned}\deg(\pi_1.\mathcal{K} + M) &= \deg(M + \pi_1.\mathcal{K}) = \deg(\mathcal{K}|M) = \deg(M|\mathcal{K}) \\ &= \deg(\mathsf{new}\, a\, \mathcal{K}) = \deg(!\mathcal{K}) = \deg(\mathcal{K}) + 1\end{aligned} \tag{3.35}$$

The purpose of a context expression is to be able to insert any process expression at the $[\,]$ place and end up with a well-formed process expression. More formal:

Definition 3.12 Context application *is defined as the mapping* $.[.] : \mathcal{PC}^\pi \times \mathcal{P}^\pi \to \mathcal{P}^\pi$ *for the empty context as* $[\,][P] = P$ *and for all other contexts recursively as follows:*

$$[\,][P] = P \qquad (\pi.\mathcal{K})[P] = \pi.\mathcal{K}[P] \tag{3.36}$$

$$(\pi.\mathcal{K} + M)[P] = \pi.\mathcal{K}[P] + M \qquad (M + \pi.\mathcal{K})[P] = M + \pi.\mathcal{K}[P] \tag{3.37}$$

$$(\mathcal{K}|Q)[P] = \mathcal{K}[P]|Q \qquad (Q|\mathcal{K})[P] = Q|\mathcal{K}[P] \tag{3.38}$$

$$(\mathsf{new}\, a\, \mathcal{K})[P] = \mathsf{new}\, a\, \mathcal{K}[P] \qquad (!\mathcal{K})[P] = !(\mathcal{K}[P]) \tag{3.39}$$

Context application is overloaded to be applied on process contexts also: $.[.] : \mathcal{PC}^\pi \times \mathcal{PC}^\pi \to \mathcal{PC}^\pi$*. Its semantic should be self-evident and will satisfies the following:*

$$\deg(\mathcal{K}) = j, \deg(\mathcal{K}') = j' \Rightarrow \deg(\mathcal{K}[\mathcal{K}]) = j + j' \tag{3.40}$$

$$(\mathcal{K}[\mathcal{K}'])[P] = \mathcal{K}[\mathcal{K}'[P]] \tag{3.41}$$

A process expression P contains usually more structure than we are interested in, e.g., we would like to consider $P|Q$ and $Q|P$ as equivalent in some sense. This is usually done by partitioning the set $\mathcal{PC}^\pi$ by the means of *congruence relations*. In general a congruence relation, $\cong$, is an equivalence relation that is preserved through algebraic operations, (e.g. if $x \cong y$ then $f(x) \cong f(y)$ for all x, y and f (with any arity)). To be more specific, we define:

Definition 3.13 (Process congruence) *An arbitrary equivalence relation* $\cong$ *is a* process congruence *if for all* $P, Q \in \mathcal{P}^\pi$ *and* $\epsilon \in \mathcal{E}$ *such that if* $P \cong Q$ *then* $\epsilon[P] \cong \epsilon[Q]$*.*

That a process congruence holds over any context application follows from:

Proposition 3.14 *For an arbitrary equivalence relation* $\cong$ *the following two statements are equivalent:*

$$\cong \textit{ is a process congruence} \tag{3.42}$$

$$\forall \mathcal{K} \in \mathcal{PC}^\pi, P, Q \in \mathcal{P}^\pi : (P \cong Q \Rightarrow \mathcal{K}[P] \cong \mathcal{K}[Q]) \tag{3.43}$$

Proof Suppose 3.43. Then it must also be true for $\mathcal{K} \in \mathcal{E}$ (since $\mathcal{E} \subset \mathcal{PC}^\pi$). Thus, $\cong$ is by definition a process congruence.

So, assume that $\cong$ is a process congruence. We shall prove 3.43 by induction on the syntactic complexity of $\mathcal{K}$.

Base case: $\deg(\mathcal{K}) = 0$. Then $\mathcal{K} \in \mathcal{E}$ and it follows from the definition of process congruence that $\forall P, Q \in \mathcal{P}^\pi : (P \cong Q \Rightarrow \mathcal{K}[P] \cong \mathcal{K}[Q])$

Induction step: Suppose that $\forall P, Q \in \mathcal{P}^\pi : (P \cong Q \Rightarrow \mathcal{K}[P] \cong \mathcal{K}[Q])$ is true for all $\mathcal{K}$ such that $\deg(\mathcal{K}) < n$ and let $\mathcal{K}'$ be arbitrary such that $\deg(\mathcal{K}') = n > 0$. There are several cases, but the proof will be identical for each one. We present only one here, namely suppose $\mathcal{K}' = \pi.\mathcal{K} + M$, and let $P, Q \in \mathcal{P}^\pi$ be arbitrary such that $P \cong Q$. As $\deg(\mathcal{K}) < n$ we know from the induction hypothesis that $\mathcal{K}[P] \cong \mathcal{K}[Q]$. Further

$$\mathcal{K}'[P] = (\pi.\mathcal{K} + M)[P] = \pi.\mathcal{K}[P] + M = (\pi.[\,] + M)[\mathcal{K}[P]] \tag{3.44}$$

$$\mathcal{K}'[Q] = (\pi.\mathcal{K} + M)[Q] = \pi.\mathcal{K}[Q] + M = (\pi.[\,] + M)[\mathcal{K}[Q]] \tag{3.45}$$

and as $\pi.[\,]+M$ is an elementary context, it follows from the definition of process context that $(\pi.[\,] + M)[\mathcal{K}[P]] \cong (\pi.[\,] + M)[\mathcal{K}[Q]]$ or $\mathcal{K}'[P] \cong \mathcal{K}'[Q]$. Induction step is completed, and proof also. □

A particular process congruence is needed. We shall define it constructively, and need the following basics first:

Definition 3.15 *The basic structural relation, $\rightleftarrows$, is defined over the set $\mathcal{P}^\pi$ of process expressions as follows. For all $P, Q \in \mathcal{P}^\pi$ and $a, a', x, y, y' \in \mathcal{N}$,*

$$\mathsf{new}\, a\, P \rightleftarrows \mathsf{new}\, a'\, \{a'/a\}P \qquad \textit{where } a' \notin \mathrm{fn}(P) \tag{3.46}$$

$$x(y).P \rightleftarrows x(y').\{y'/y\}P \qquad \textit{where } y' \notin \mathrm{fn}(P) \tag{3.47}$$

$$P + Q \rightleftarrows Q + P \tag{3.48}$$

$$P + (Q + R) \rightleftarrows (P + Q) + R \tag{3.49}$$

$$P|0 \rightleftarrows P \tag{3.50}$$

$$P|Q \rightleftarrows Q|P \tag{3.51}$$

$$P|(Q|R) \rightleftarrows (P|Q)|R \tag{3.52}$$

$$\mathsf{new}\, x\, (P|Q) \rightleftarrows P|\, \mathsf{new}\, x\, Q \qquad \textit{if } x \notin \mathrm{fn}(P) \tag{3.53}$$

$$\mathsf{new}\, x\, 0 \rightleftarrows 0 \tag{3.54}$$

$$\mathsf{new}\, xy\, P \rightleftarrows \mathsf{new}\, yx\, P \tag{3.55}$$

$$!P \rightleftarrows P|\, !P \tag{3.56}$$

Further, the equations work both ways ($\rightleftarrows$ is commutative.)

We could now define structural congruence as the least congruence relation that satisfies the above equations, or we can say that two process expressions are structural congruent if we can transform one into the other by manipulating subexpressions by using $\rightleftarrows$ repeatedly. However, we choose to forward constructively.

As subexpressions are not taken care of we see that even if $P|0 \rightleftarrows P$, it is still such that $P|0 + Q \not\rightleftarrows P + Q$. The next operation is to define a relation that takes care of subexpressions:

Definition 3.16 *The relation $\equiv^n$ is defined over the set $\mathcal{P}^\pi$ as follows:*

$$P \equiv^i Q \text{ if and only if } \exists \mathcal{K} \in \mathcal{PC}^\pi, P', Q' \in \mathcal{P}^\pi : P = \mathcal{K}[P'] \wedge \quad (3.57)$$
$$Q = \mathcal{K}[Q'] \wedge \quad (3.58)$$
$$\deg(\mathcal{K}) = i \wedge P' \rightleftarrows Q' \quad (3.59)$$

Example:

$$P = R + \bar{x}\langle\rangle.(S|0) \qquad Q = R + \bar{x}\langle\rangle.S \quad (3.60)$$
$$\mathcal{K} = R + \bar{x}\langle\rangle.[\,] \qquad \deg(\mathcal{K}) = 2 \quad (3.61)$$
$$P' = (S|0) \qquad Q' = S \quad (3.62)$$
$$P' \rightleftarrows Q' \quad (3.63)$$
$$\text{Therefore: } P \equiv^2 Q \quad (3.64)$$

Proposition 3.17 *$\equiv^n$ is symmetric.*

Proof Follows easily from the fact that $\rightleftarrows$ is symmetric. □

What we in fact has defined is a class of relations, i.e. $\{\equiv^i\}_{i \in N}$. However, neither of them is reflexive nor transitive. E.g. for every $i \in N$ it is so that $0 \not\equiv^i 0$ and also:

$$P = x.(S|0) + y.(!T|U) \quad (3.65)$$
$$Q = x.S + y.((T|!T)|U) \quad (3.66)$$
$$P \not\equiv^i Q \quad (3.67)$$

Thus, we need to extend $\equiv^n$ reflexively and transitively:

Definition 3.18 *The relation $\equiv_n$ is defined over the set $\mathcal{P}^\pi$ as follows:*

$$P \equiv_0 Q \text{ if and only if } P = Q \quad (3.68)$$
$$P \equiv_{i+1} Q \text{ if and only if } \exists R \in \mathcal{P}^\pi, j : P \equiv_i R \wedge R \equiv^j Q \quad (3.69)$$

Considering the example above:

$$R_1 = P \qquad R_2 = x.(S|0) + y.((T|!T)|U) \quad (3.70)$$
$$j_1 = 3 \qquad j_2 = 2 \quad (3.71)$$
$$R_1 \equiv^3 R_2 \qquad R_2 \equiv^2 Q \quad (3.72)$$
$$(3.73)$$

Finally, structural congruence can be defined:

Definition 3.19 *The relation $\equiv$ is defined over the set $\mathcal{P}^\pi$ as follows:*

$$P \equiv Q \text{ if and only if } \exists i : P \equiv_i Q$$

It remains to show that $\equiv$ is a process congruence:

Proposition 3.20 *The relation $\equiv$ is a process congruence.*

Proof We must show that $\equiv$ is an equivalence relation, and that it is preserved over elementary contexts.

That $\equiv$ is reflexive follows from the fact that $P \equiv_0 P$ for every $P \in \mathcal{P}^\pi$. Transitivity from that if $P \equiv Q$ and $Q \equiv R$, then there exists i and j such that $P \equiv_i Q$ and $Q \equiv_j R$, thus with a simple induction argument we have that $P \equiv_{i+j} R$. Symmetricness follows also easily from that $\equiv^n$ is symmetric.

So what remains is to show that $\equiv$ satisfies the condition of definition 3.13.

Suppose $P \equiv Q$, we must prove that for every elementary process context $\epsilon \in \mathcal{PC}^\pi$, we have that $\epsilon[P] \equiv \epsilon[Q]$.

This is done by double induction, first on the length of a $\equiv_n$-chain and on the depth of a $\equiv^n$ relation. Let ϵ be an arbitrary elementary context.

Base case: In this case we have that $P \equiv_0 Q$ which means that $P = Q$, and trivially $\epsilon[P] \equiv \epsilon[Q]$.

Induction step: Suppose for all P, Q and $i < n (n > 0)$ such that if $P \equiv_i Q$ we have that $\epsilon[P] \equiv_i \epsilon[Q]$ (induction hypothesis). Let P and Q be arbitrary such that $P \equiv_n Q$. Then there exists an R and a j such that $P \equiv_{n-1} R$ and $R \equiv^j Q$. The induction hypothesis implies that $\epsilon[P] \equiv_{n-1} \epsilon[R]$ so if we can show that

$$\forall j\colon\ R \equiv^j Q \Rightarrow \epsilon[R] \equiv^j \epsilon[Q] \tag{3.74}$$

we can use transitivity to conclude that $\epsilon[P] \equiv_n \epsilon[Q]$ which thereby finishes the induction step, and the proposition follows directly.

Proof of 3.74: We want to prove that for all $R \equiv^j Q$, $\epsilon[R] \equiv^j \epsilon[Q]$ follows. If the premise holds, we know that there exists a process context $\mathcal{K} \in \mathcal{PC}^\pi$, and process expressions $R', Q' \in \mathcal{P}^\pi$ such that $j = \deg(\mathcal{K}), R' \rightleftarrows Q', R = \mathcal{K}[R']$ and $Q = \mathcal{K}[Q']$.

However, by definition the expression $\epsilon[\mathcal{K}]$ is a process context of degree $j + 1$. Thus, we automatically have that $(\epsilon[\mathcal{K}])[R'] \equiv^{j+1} (\epsilon[\mathcal{K}])[Q']$, or that $\epsilon[\mathcal{K}[R']] \equiv^{j+1} \epsilon[\mathcal{K}[Q']]$ from which 3.74 follows. □

3.1.3 Syntactic sugar

This section introduces some useful syntactic sugar, that is, high-level notation that can be compiled into the core π-calculus. Unlike Milner we do not compile all name binding operations down to abstraction.

The polyadic π-calculus

As the observant reader will have noticed the core calculus of Definition 3.2 allows only processes to communicate one name (or value) at a time over a channel. Instead of giving a semantics to multiple arguments in the core calculus the following translation into the core ensure that multiple argument syntax is well-defined:

$$x(y_1, \ldots, y_n).P \Rightarrow x(w).w(y_1).\cdots.w(y_n).P$$
$$\bar{x}\langle z_1, \ldots, z_n\rangle.Q \Rightarrow \mathsf{new}\, w\, x(w).w(z_1).\cdots.w(z_n).Q$$

We leave it as an exercise to the interested reader to show that the 'obvious translation' is to simple—it can lead to mix-up in the presence of more two processes that communicate on a channel.

Process definition and application

A *process definition* is meta-syntax for introducing a name for a process expression. The names introduce are called *process identifiers*; we shall denote them as A, B, P, Q.

In general process definitions can be parametric, that is, a definition may require arguments—that is names—to be supplied when the it is referenced in process expression. Those arguments will then be bound to the parameter names when the process is used.

Definition of process identifier A with distinct parameters $x_1, \ldots, x_n$ and body Q_A:

$$A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} Q_A.$$

When referring to a process identifier—such as A above—outside of a process expression we sometimes write $A : n$ to indicate that A has n arguments. Also, with such a definition in place we abuse terminology and refer to the 'process A'. Using the process A on arguments $y_1, \ldots, y_n$ is denoted as follows.

$$A\langle y_1, \ldots, y_n \rangle.$$

For recursive process definitions things get more complicated. A recursive process is the process resulting from a definition where a process identifier A occurs also on the right-hand side of the '$\stackrel{\text{def}}{=}$' sign, for example,

$$A(x, y) \stackrel{\text{def}}{=} \bar{x}\langle y \rangle.A\langle y, x \rangle\,.$$

Given a recursive definition of an identifier

$$A \stackrel{\text{def}}{=} Q_A,$$

that is Q_A refers to A, and where the scope of this definition is some process P. Here is Milner's procedure [25, Ch. 9.5] for 'compiling' the right-hand side P into a process expression in the core syntax π-calculus of Definition 3.2.

1. Invent a new name, say a, to stand for A.
2. Let $\widehat{R}$ denote the result of replacing every application $A\langle w_1, \ldots, w_n \rangle$ by $\bar{a}\langle w_1, \ldots, w_n \rangle$ in process R.
3. Replace P, and the accompanying definition of A by the following core process expression:

$$\mathsf{new}\, a\, (\widehat{P} | !a(x_1, \ldots, x_n).\widehat{Q_A}\,.$$

The generalization to mutually recursive process definitions in straightforward.

3.1.4 Reactions

The rules for reactions gives us a recursive definition of the reaction-trees.Since there is no branching rule we might call these objects *reaction-truncs*. Given two expressions $P, Q \in \mathcal{P}^\pi$, we call $P \longrightarrow Q$ a *reaction sequent*,and we call P and Q respectively the *source* and *target* of a reaction sequent.

Definition 3.21 (Reaction rules)

$$\tau.P + M \longrightarrow P \qquad \text{TAU}$$

$$(x(y).P + M)|(\bar{x}\langle y\rangle.Q + N) \longrightarrow \{z/y\}P|Q \qquad \text{REACT}$$

$$\frac{\begin{array}{c}\Pi\\ |\\ P \longrightarrow P'\end{array}}{P|Q \longrightarrow P'|Q}\ \text{PAR} \qquad \frac{\begin{array}{c}\Pi\\ |\\ P \longrightarrow P'\end{array}}{\mathsf{new}\, a\, P \longrightarrow \mathsf{new}\, a\, P'}\ \text{RES}$$

$$\frac{\begin{array}{c}\Pi\\ |\\ P \longrightarrow P' \quad P \equiv Q \quad P' \equiv Q'\end{array}}{Q \longrightarrow Q'}\ \text{STRUCT}$$

In the definition above *immediate sub-trunc* is defined implicitly, in the obvious way: TAU, REACT has no immediate sub-trunk, PAR, RES and STRUCT has the immediate subproof

$$\begin{array}{c}\Pi\\ |\\ P \longrightarrow P'\end{array}$$

We shall write $\Pi \vdash P \longrightarrow P'$ meaning that there exists a derivation (reaction-trunk) Π for the reaction-sequent $P \longrightarrow P'$.

Definition 3.22 (Depth of reaction-truncs) *$d(\Pi)$ is defined by:*

$$d(\text{TAU}) = d(\text{REACT}) = 0 \tag{3.75}$$

$$\begin{array}{l}\textit{If the last rules in } \Pi \textit{ is } \text{PAR}, \text{RES} \textit{ or } \text{STRUCT} \textit{ with}\\ \qquad \textit{immediate sub-trunk } \Pi_1 \textit{ then } d(\Pi) = d(\Pi_1) + 1\end{array} \tag{3.76}$$

We observe that the struct-rule is a genuinely pseudo-rule, highly non-constructive. It carries all the information of structural congruence inside. From a computational point of view this is not nice. A way to interpret this is that the formal calculus is infected by the semantics of the calculus. Given a reaction sequent, the obvious recursive construction of the reaction-trunk from bottom up fail in case of the struct-rule, where the theorem-prover must take a brake and run through every process-expression structurally equivalent to Q and Q', which will take infinite time for both Q and Q'. First, some definitions:

Definition 3.23 *The reflexive and transitive closure of $\longrightarrow$:*

$$P \xrightarrow{*} P \tag{3.77}$$

$$P \xrightarrow{*} Q \textit{ iff } \exists R(P \xrightarrow{*} R \wedge R \longrightarrow Q) \tag{3.78}$$

3.1.5 Time and computation

A natural question to pose is what role time does play in a calculus that is modeling synchronous communication. As we shall see later, when giving formal models of concrete contexts of communication, asynchronicity drops in the back-door every moment.

Time comes into play at both an abstract and a concrete level. First, on the level of language-abstraction it is important to have a recursive grounding

of the number of rewriting-steps necessary to show the structural equivalence of two $\mathcal{P}^{\pi}$. This was taken care of in the previous section. Second, on the level of concrete transactions of a concrete process-expression, it is important to understand how concepts of time are apparent.

There is no discussion of 'time' in [25]. Time occurs in the ambient-calculus of [6], and in the π-calculus in [27]. In both papers temporal concepts are connected to relations of transitions (actions). In [27] modality is introduced to reason about bisimilarity. "It may be shown that two processes are strongly bisimilar iff they satisfy the same formulae (...)" [1]. The main goal of Milner *et al.* is to study the high level specification languages capturing and unifying different formulations of (structural) equivalences. But, the expressibility of the specification language of time in π-calculus is rather limited. Let us recapitulate Milner, Parrow and Walker [27]:

$$P \models \mathbf{true} \tag{3.79}$$

$$P \models \neg A \text{ iff } P \not\models A \tag{3.80}$$

$$P \models A \wedge B \text{ iff } P \models A \text{ and } P \models B \tag{3.81}$$

$$P \models \langle \alpha \rangle\ A \text{ iff } \exists P'(P \xrightarrow{\alpha} P' \wedge P' \models A) \tag{3.82}$$

Note that α is an action prefix. The interpretation of formulas are given by the standard Tarski semantics, with the only atomary formula $P \models \mathbf{true}$. But the expressibility of the logical language itself is rather poor, actually the only structure uncovered are reactions.

In [6], the corresponding formulas of the logical language is containing the full flavour of processes. That is, the properties of the ambients are mapped into the logical language. The temporal possibility modality is explicated as the reduction relation on ambients. It is a proper "extention" of the reaction relation of Milner. There are some differences though. The ambient calculus is restriction-free and do not have the ordinary reaction rule. Cardelli and Gordons reductions describes the dynamic behaviour of ambients.

$$n[in\ m.P|Q]|m[R] \longrightarrow m[n[P|Q]|R] \qquad (RedIn)$$

$$m[n[out\ m.P|Q]|R] \longrightarrow n[P|Q]|m[R] \qquad (RedOut)$$

$$open\ n.P|n[Q] \longrightarrow P|Q \qquad (RedOpen)$$

$$(n).P|\langle M \rangle \longrightarrow \{M/n\}P \qquad (RedComm)$$

$$\text{If } P \longrightarrow Q \text{ then } n[P] \longrightarrow n[Q] \qquad (RedAmb)$$

$$\text{If } P \longrightarrow Q \text{ then } P|R \longrightarrow Q|R \qquad (RedPar)$$

$$\text{If } P \longrightarrow Q, P \equiv P', Q \equiv Q' \text{ then } P' \longrightarrow Q' \qquad (Red \equiv)$$

But (Red Comm) can easily be proved using the react axiom of Milner:
$(x(y).P + M)|(\bar{x}\langle y \rangle.Q + N) \longrightarrow \{z/y\}P|Q$ has the conrete instance:
$(x(y).P)|(\bar{x}\langle m \rangle.0) \longrightarrow \{m/y\}P|0 \equiv \{m/y\}P$, letting m denote M.

Cardelli and Gordon's specification logic bimodal capturing the notions of both time and place. In case of time we cite:

$$P \models \diamond A \text{ iff } \exists P'(P \xrightarrow{*} P' \wedge P' \models A)$$

which means that A is realizable once upon a time in the future.

[1] p.150.

Since the diamond and box operators of modal logic are defined by quantifiers we are only capable of expressing abstract properties of time, like "sometimes", "eventually", "always" and so on.

We can make a twist to the problem a start searching for temporality inside π-calculus. The natural starting point is *reaction*, since the source and target in a reaction are two perspectives on an interaction between processes, with an implicit reference to time, referring to respectively t_1 and t_2, two points in an execution-graph of processes. Given a process expression in π-calculus, we can form the graph of reaction sequents, we call this a *reaction graph*. A time-interval is a path in a reaction-graph.

Definition 3.24 *Let $Q \in \mathcal{P}^\pi$, then the reaction-graph of Q called $rg(Q)$ is the set of reactions such that $rg(Q) = \{Q_1 \longrightarrow Q_2 | Q \xrightarrow{*} Q_1\}$*

We note that by α-conversion and the fact that $P|0 \rightleftarrows P$ (basic rewriting rules), we might immediately get ω many react-elements, by STRUCT.

Definition 3.25 *A reaction $P \longrightarrow Q$ is* conservative *iff $deg(P) \leq deg(Q)$ and* $\mathrm{fn}(Q) \subseteq \mathrm{fn}(P)$

Definition 3.26 *A reduction graph Γ is* conservative *iff every reaction $P \longrightarrow Q \in \Gamma$ is conservative.*

Proposition 3.27 *If $Q \in \mathcal{P}^\pi$ is not containing replication, then there exists a conservative reaction graph $rg(Q)$ such that $|rg(Q)| \leq n$.*

Proof Must be fixed. □

Proposition 3.28 *If $Q \in \mathcal{P}^\pi$ contains replication, then $|rg(Q)| = \omega$.*

Proof Must be fixed. □

3.2 Specification

π-calculus is often considered as a specification language by its own. With this, one can express many basic ideas involving communication and parallelism and such. However, one find sometimes the need for a still more abstract language to express what the communication protocols expressed in π-calculus really try to achieve. That is, a specification of π-calculus.

Ambient calculus [..] which is built upon π-calculus has both an formal object language and a specification language which uses modal logic in order to express properties with ambient programs. We do not have such tools in π-calculus, so we try to develop some ideas based on SPI calculus [1].

3.2.1 Barbing

Consider any channel x and process expression P. From [1] we borrow the notation $P \downarrow x$ to express that the process P is able to receive a message on channel x. We write $P \downarrow \overline{x}$ to express that the process P is able to send a message on channel x. If we write $P \downarrow \beta$ to mean either one.

Formally, this is defined as:

Definition 3.29 (Barbing)

$$\bar{x}\langle y\rangle.P\downarrow\overline{x} \qquad x(y).P\downarrow x$$
$$\frac{P\downarrow\beta}{(P+Q)\downarrow\beta} \qquad \frac{P\downarrow\beta}{\mathsf{new}\, m\, P\downarrow\beta} \quad \beta\notin\{m,\overline{m}\}$$
$$\frac{P\downarrow\beta}{P|Q\downarrow\beta} \qquad \frac{P\equiv Q,\ P\downarrow\beta}{Q\downarrow\beta}$$

This is the position just now. [1] extended this into the future:

Definition 3.30 (General barbing)

$$\frac{P\downarrow\beta}{P\Downarrow\beta} \qquad \frac{P\longrightarrow Q,\ Q\Downarrow\beta}{P\Downarrow\beta}$$

That is, $P\Downarrow\beta$ if there is some reaction chain from P that leads into a process that is able to communicate on β.

3.2.2 Testing equivalence

[1] goes further and defines a *test* that is a pair $\langle R,\beta\rangle$ of a process expression R and a channel (input or output) β such that we say that a process P *passes* the test if $(P|R)\Downarrow\beta$. Two processes may now be compared:

Definition 3.31 (Testing equivalence)

$$P\sqsubseteq Q \stackrel{\text{def}}{=} \forall\langle R,\beta\rangle\in\mathcal{P}^{\pi}\times\mathcal{N}\colon\ (P|R)\Downarrow\beta\Rightarrow(Q|R)\Downarrow\beta$$
$$P\leftrightsquigarrow Q \stackrel{\text{def}}{=} P\sqsubseteq Q\wedge Q\sqsubseteq P$$

That is $P\leftrightsquigarrow Q$, i.e. are *testing equivalent*, whenever P passes a test Q does also and vice versa. (There is a small irregularity. The channel name β is said to be either an input or an output channel. However, channel names are not categorized such. In reality, we are not talking about the set of names $\mathcal{N}$, but a set of *tagged* names: $\mathcal{N}\times\{\text{in},\text{out}\}$.)

So far, everything is taken from [1].

3.2.3 Some notation

Given any preorder (reflexive and transitive) relation $\preceq$. We extend this on sets:

$$A\preceq B \stackrel{\text{def}}{=} \forall a\in A\,\exists b\in B\colon\ a\preceq b$$

We do the same with equivalence relations, $\approx$, however, here we must ensure that it works both ways. That is, given any equivalence relation $\approx$:

$$A\approx B \stackrel{\text{def}}{=} (\forall a\in A\,\exists b\in B\colon\ a\approx b)\wedge(\forall b\in B\,\exists a\in A\colon\ b\approx a)$$

It is simple to show that the preorder on sets is a preorder, and that equivalence on sets is an equivalence relation.

Barbing and generalized barbing are also extended to sets.

$$A\downarrow\beta \stackrel{\text{def}}{=} \forall a\in A\colon\ a\downarrow\beta$$
$$A\Downarrow\beta \stackrel{\text{def}}{=} \forall a\in A\colon\ a\Downarrow\beta$$

3.2.4 Invariants

General barbing is existential in nature. That is, $P \Downarrow \beta$ if there *exists* a reaction path that ends in a situation such that the final process is able to participate in a communication on channel β. We would also like to express properties that this is a necessity.

Define the reaction set, *rs* to be the set of process expressions that might follow from a given P:

Definition 3.32 (Reaction set)

$$\mathrm{rs}(P) \stackrel{\mathrm{def}}{=} \{Q | P \stackrel{*}{\longrightarrow} Q\}$$

If a process cannot do anything by its own, it is terminal:

Definition 3.33 (Terminal process) *A process expression P is **terminal** if $\{P\} \equiv \mathrm{rs}(P)$. The set of terminal process expression originating from P is denoted Term, that is: $Term(P) = \{Q | Q \in \mathrm{rs}(P) \wedge Q$ is terminal$\}$.*

It is essential terminal if, whatever the process does we cannot observe it:

Definition 3.34 (Essential terminal) *A process expression P is **essential terminal** if $\{P\} \leftrightsquigarrow \mathrm{rs}(P)$*

A process expression that constantly is able to participate on a communication channel x is called x-stable:

Definition 3.35 (Stable) *A process expression P is called β-**stable** if $\mathrm{rs}(P)\downarrow \beta$.*

It might happen that a process expression sometimes is busy doing other work, but that is eventually will be able to do some communication on a given channel. We call this forever x-able:

Definition 3.36 (Forever able) *A process expression is **forever** β-**able** if $\mathrm{rs}(P) \Downarrow \beta$*

In other cases we want the process to eventually settle down in a stable listening situation:

Definition 3.37 *A process expression is **eventually** β-**stable** if the set $S = \{Q | Q \in \mathrm{rs}(P), \neg Q \downarrow \beta\}$ is finite and $S \cap Term(P) = \emptyset$.*

That is, if the process is in any state where it is not capable to communicate on channel β this won't last long and it is certainly not terminal.

3.2.5 Liveness

We are now able to express some liveness properties. A process is able to forever listen on a channel x if it receives a request, handles it, and settles down listening on the same channel:

Definition 3.38 (Single input able) *A process is **Single x-input stable** if $\forall y\colon (\bar{x}\langle y\rangle.0 | P)$ is eventually x-stable.*

Note that x here really denotes the input end of the channel. However, we want this to last forever:

Definition 3.39 (Forever input able) *A process is **Forever x-input able** if $(!\,\mathsf{new}\, y\, \bar{x}\langle y\rangle.0) | P)$ is forever x-able.*

Chapter 4

Application: Modeling communicating systems

In this chapter we apply formal techniques to analyze communicating systems in a fundamental way—what are they and how do they behave?

Our method is conceptual analysis: we give English-language descriptions of communication acts, and then we formalize the concepts used there with the machinery of the π-calculus. The calculus allows us to build models of the acts and give them precise meaning based on the meaning of π-calculus expressions.

The method lets us uncover fundamental properties of the communicating systems we study, and also identifies strengths and weaknesses of our chosen formalism—the π-calculus itself.

4.1 Preliminaries

An *actor* A is a process definition:

$$A(x_1, \ldots, x_n) \stackrel{\text{def}}{=} P_1 | \cdots | P_m,$$

where arguments $x_1, \ldots, x_n$ can be considered parameters to the actor, and the processes $P_1, \ldots, P_m$ are a logically related set of processes.

We denote actors by A, B.

4.2 Modeling basic communication

This section studies basic communication systems and acts. In each case there are two communicating actors and neither 'moves around' the network.

We stick to the following simple description of an act of communication where the static part describes the initial system and the dynamic part describes the system's behavior.

Act 4.1 *Static part: There are actors A, B and a network connecting them somehow. The actors reside in the same place of the network at all times. Both actors have access to a resource n providing them with the capacity to communicate with each other across the network.*

Dynamic part: Actor A communicates data x to actor B across the network.

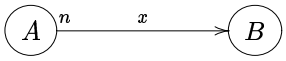


Figure 4.1: Direct communication over fixed channel n in Model 4.2.

The following is the simplest model for Act 4.1 possible in the π-calculus.

Model 4.2 *Actors:*

$$A(n) \stackrel{\text{def}}{=} \bar{n}\langle x\rangle.A_1 \qquad\qquad B(n) \stackrel{\text{def}}{=} n(y).B_1$$

Communication:

$$A\langle n\rangle|B\langle n\rangle = \underline{\bar{n}\langle x\rangle}.A_1|\underline{n(y)}.B_1 \longrightarrow A_1|\{x/y\}B_1$$

This is the only reaction sequence possible for the system $A\langle n\rangle|B\langle n\rangle$; it is depicted in Figure 4.1. In our diagram notation an arrow represents the sending of data (here x) across a named resource (here channel n).

Remarks:

- Data communication is 'ideal': it's synchronous, instantaneous, no data is lost or corrupted, and data elements is always sent and received in the same order. (Transfer of several data elements $x_1, \ldots, x_m$ could be realized using multiple arguments, $n\langle x_1, \ldots, x_m\rangle$.)
- The resource providing communication capacity is modeled simply by a fixed π-calculus channel that both actors get knowledge about initially.

The model does not compare very well to our intuitions about the communication act above. Especially the 'network' part of the model seems quite primitive.

Here's another model where we introduce a separate entity N to model the network, and let N provide a fresh channel on which communication between A and B is to take place. The entity N is contacted on a fixed channel n.

Model 4.3 *Actors:*

$$A \stackrel{\text{def}}{=} n(a).\bar{a}\langle x\rangle.A_1 \qquad\qquad B \stackrel{\text{def}}{=} n(b).b(y).B_1$$

$$N \stackrel{\text{def}}{=} \mathsf{new}\, d\,(\bar{n}\langle d\rangle|\bar{n}\langle d\rangle)$$

where we assume that $d \notin \mathrm{fn}(A\langle n\rangle|B\langle n\rangle)$.[1]

Communication: See Figure 4.2. In the figure A_2, B_2 *are renamings of* A_1, B_1 *due to the name bound by input on* n.[2]

In the above reduction sequence Equation (t_0) models that A accesses the network, N, to obtain a communication resource, and Equation (t_1) models that B does the same. The actual data communication between A and B is modeled by Equation (t_2). The diagram series in Figure 4.3 shows this, with times corresponding to π-calculus reductions.

[1] In the following we assume that a restriction $\mathsf{new}\, a\, P$ introduces a new (globally) unique name a instead of explicitly making assumptions like this.

[2] From now on we sometimes sweep under the carpet model issues that are only internal to the model and that do not mirror aspects of the act being modeled.

$$
\begin{aligned}
& A|B|N \\
& \quad = n(a).\bar{a}\langle x\rangle.A_1|n(b).b(y).B_1|\underline{\mathsf{new}\, d}\,(\bar{n}\langle d\rangle|\bar{n}\langle d\rangle) \\
& \quad \equiv \mathsf{new}\, d\,(\underline{n(a)}.\bar{a}\langle x\rangle.A_1|n(b).b(y).B_1|\underline{\bar{n}\langle d\rangle}|\bar{n}\langle d\rangle) && (t_0) \\
& \longrightarrow \mathsf{new}\, d\,(\bar{d}\langle x\rangle.A_2|\underline{n(b)}.b(y).B_1|0|\underline{\bar{n}\langle d\rangle}) && (t_1) \\
& \longrightarrow \mathsf{new}\, d\,(\underline{\bar{d}\langle x\rangle}.A_2|\underline{d(y)}.B_2|0|0) && (t_2) \\
& \longrightarrow \mathsf{new}\, d\,(A_2|\{x/y\}B_2|0|0) \\
& \quad \equiv A_2|\{x/y\}B_2|\,\mathsf{new}\, d\,(0|0)
\end{aligned}
$$

Figure 4.2: Reductions in Model 4.3.

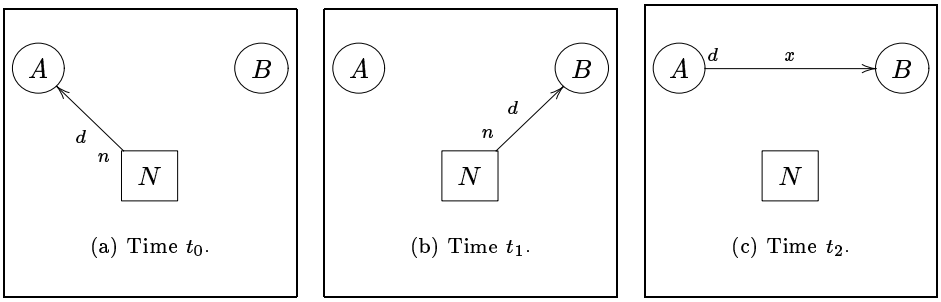


Figure 4.3: Direct communication on new channel n in Model 4.3.

The reduction sequence shown here models that A accesses the network before B does, but there is another reduction sequence for the same system that models B accessing it first. Both reduction sequences give the same end result (up to structural congruence), so the dynamic aspect of the model is in a sense robust.

Further remarks to Model 4.3:

- Communication is provided by resource n, but the actual peer-to-peer communication capacity is provided by a newly allocated π-calculus channel d. This also models the port/socket mechanisms of Unix.
- N died after helping A, B to communicate once, preventing further channels of communication to be established between the two.
- Data transfer is still ideal in the sense remarked to Model 4.2.

Note that the two models seen so far rely on the fact that A and B are the only actors trying to communicate using n. If this were not the case another actor could interfere with the communication.

The next model solves these problems by introducing a more sophisticated network. First the network process N is defined as the replication of a different process N_C, and it is once instance of the latter process that models the specific service of communication between two specific actors. N can be viewed as a pool of communication resources for pairs of actors.

Second the network allocates not just a fresh π-calculus channel, but a process C to serve as a better model of a real channel than a π-calculus channel

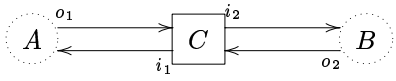


Figure 4.4: Process C modeling a thick duplex channel $C\langle i_1, o_1, i_2, o_2\rangle$.

$$
\begin{aligned}
&A\langle a,b\rangle|B\langle a,b\rangle|N \\
&\quad = \bar{n}\langle a,b\rangle.a(i,o).\bar{o}\langle x\rangle.A_1|b(i,o).i(y).B_1 \\
&\qquad |\underline{!n(c_1,c_2).\,\mathsf{new}\, i_1o_1i_2o_2\,(\bar{c_1}\langle i_1,o_1\rangle|\bar{c_2}\langle i_2,o_2\rangle|C\langle i_1,o_1,i_2,o_2\rangle)} \\
&\equiv = \underline{\bar{n}\langle a,b\rangle}.a(i,o).\bar{o}\langle x\rangle.A_1|b(i,o).i(y).B_1 \\
&\qquad |\underline{n(c_1,c_2)}.\,\mathsf{new}\, i_1o_1i_2o_2\,(\bar{c_1}\langle i_1,o_1\rangle|\bar{c_2}\langle i_2,o_2\rangle|C\langle i_1,o_1,i_2,o_2\rangle)|N \qquad (t_0) \\
&\longrightarrow a(i,o).\bar{o}\langle x\rangle.A_1|b(i,o).i(y).B_1 \\
&\qquad |\underline{\mathsf{new}\, i_1o_1i_2o_2}\,(\bar{a}\langle i_1,o_1\rangle|\bar{b}\langle i_2,o_2\rangle|C\langle i_1,o_1,i_2,o_2\rangle)|N \\
&\quad \equiv \mathsf{new}\, i_1o_1i_2o_2\,[\underline{a(i,o)}.\bar{o}\langle x\rangle.A_1|b(i,o).i(y).B_1 \\
&\qquad\qquad |\underline{\bar{a}\langle i_1,o_1\rangle}|\bar{b}\langle i_2,o_2\rangle|C\langle i_1,o_1,i_2,o_2\rangle]|N \qquad (t_1) \\
&\longrightarrow \mathsf{new}\, i_1o_1i_2o_2\,[\bar{o_1}\langle x\rangle.A_2|\underline{b(i,o)}.i(y).B_1|\mathsf{0}|\underline{\bar{b}\langle i_2,o_2\rangle}|C\langle i_1,o_1,i_2,o_2\rangle]|N \qquad (t_2) \\
&\longrightarrow \mathsf{new}\, i_1o_1i_2o_2\,[\underline{\bar{o_1}\langle x\rangle}.A_2|i_2(y).B_2|\mathsf{0}|\mathsf{0}|\underline{C\langle i_1,o_1,i_2,o_2\rangle}]|N \qquad (t_3) \\
&\longrightarrow \mathsf{new}\, i_1o_1i_2o_2\,[A_2|i_2(y).B_2|\mathsf{0}|\mathsf{0}|C'\langle i_1,o_1,i_2,o_2\rangle]|N \\
&\qquad C' \text{ does something to } x\text{; then sends it out on } i_2\text{:} \\
&\xrightarrow{*} \mathsf{new}\, i_1o_1i_2o_2\,[A_2|\underline{i_2(y)}.B_2|\mathsf{0}|\mathsf{0}|\underline{C''\langle i_1,o_1,i_2,o_2\rangle}]|N \qquad (t_4) \\
&\longrightarrow \underline{\mathsf{new}\, i_1o_1i_2o_2}\,[A_2|\{x/y\}B_2|\underline{\mathsf{0}|\mathsf{0}}|C'''\langle i_1,o_1,i_2,o_2\rangle]|N \\
&\quad \equiv A_2|\{x/y\}B_2|\,\mathsf{new}\, i_1o_1i_2o_2\,[C'''\langle i_1,o_1,i_2,o_2\rangle]|N
\end{aligned}
$$

Figure 4.5: Reductions in Model 4.4.

does. We call models of channels that involve π-calculus process 'thick channels'. The process $C : 4$ takes 4 arguments i_1, o_1, i_2, o_2 and makes a duplex channel process with two inputs i_1, i_2 and two outputs o_1, o_2. What goes in on i_1 comes out on o_2 and vice versa for i_2, o_1. Such a channel, from a black box viewpoint, is shown in Figure 4.4. The next model also addresses other problems.

Model 4.4 *Actors:*

$$
\begin{aligned}
A(a,b) &\stackrel{\text{def}}{=} \bar{n}\langle a,b\rangle.a(i,o).\bar{o}\langle x\rangle.A_1 \\
B(a,b) &\stackrel{\text{def}}{=} b(i,o).i(y).B_1 \\
N &\stackrel{\text{def}}{=} !N_C \\
N_C &\stackrel{\text{def}}{=} n(c_1,c_2).\,\mathsf{new}\, i_1o_1i_2o_2\,(\bar{c_1}\langle i_1,o_1\rangle|\bar{c_2}\langle i_2,o_2\rangle|C\langle i_1,o_1,i_2,o_2\rangle)
\end{aligned}
$$

Communication: See Figure 4.5.

Note that both actors are identified by a unique π-calculus channel name, A by a and B by b, and that both have knowledge about their own name and the

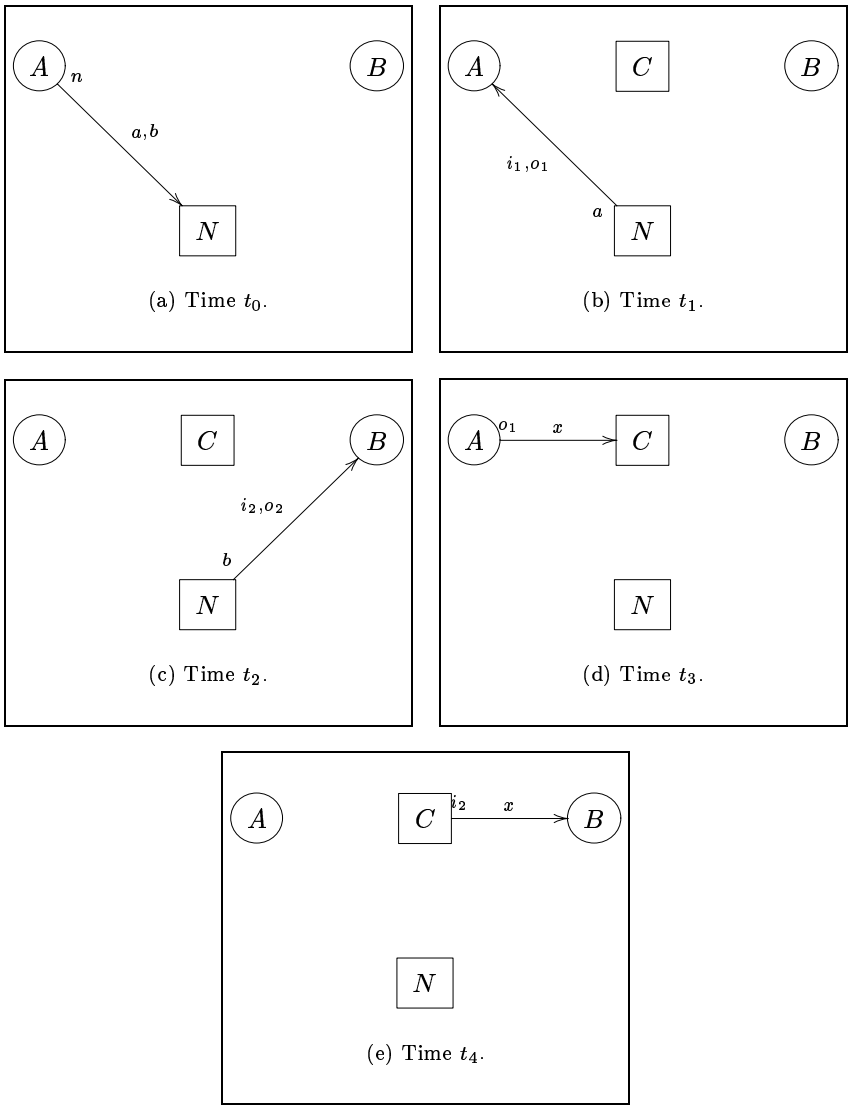


Figure 4.6: Communication on thick channel C in Model 4.4.

other actor's name. For example the actors can be created like this:

$$\mathsf{new}\, ab\, (A\langle a, b\rangle | B\langle a, b\rangle).$$

This convention is necessary if both actors are to be able to initiate communication with the other, and it is a reasonable model: an actor can't communicate with somebody it cannot identify.

As before, the reduction sequence shown in Figure 4.5 is just one possibility but all results are the same up to structural congruence. Figure 4.6 shows diagrammatically what happens in the reduction sequence:

(t_0): A requests a duplex channel from the network. The channel should connect A itself—identified by a and B—identified by b.

(t_1): N allocates a thick duplex channel C, and A gets a fresh pair of I/O channels connected to C back from N.

(t_2): N sends B a different pair of fresh I/O channels, also connected to C.

(t_3): Using its output channel o_1 actor A sends data x into C, where x is in transit.

(t_4): Eventually C sends data x to B using B's input channel i_2.

Remarks to Model 4.4:

- We choose to say that C is allocated the moment once N communicates one of its π-calculus channel names to an actor.
- Each time it is contacted process N will create a new thick channel and communicate its π-calculus channel names to the actors involved. This ensures that channels between different actors cannot get interfere with each other.
- Communication is no longer necessarily ideal: the 'thick channel' C can be made to delay messages (for some reduction steps) or re-order them using a local buffer. Non-deterministic data loss can be realized by including "$+0$" at some point in C where data is passed around.

The model is not perfect, for example, the actors always gets a resource (channel name) from the network. In a real system obtaining this resource would sometimes fail.

A final subtle point about the model. The thick channel C models a network, and naturally it would be used model that communication across the network takes time. But assume that channel x identifies a local resource R_A inside A. When A sends x to B using C, then B will have access to R_A is just the same manner that A does, and this is not a natural model. One way to avoid this would be to introduce a more sophisticated thick channel C that would translate x to x' and send x' to B, thereby trapping B's communication on x' and delaying it before sending it out on x.

4.3 Modeling channels

This section explores how channels and patterns of communication can be modeled using the π-calculus.

4.3.1 Channel characteristics

The simplest form of channel we have in our model language is the π-calculus channel. Generalization to the polyadic case, that is, several arguments, can be done using the standard 'trick' used for π-calculus channels in Section 3.1.3.

Here are some simple channels of varying characteristics.

Perfect simplex channel:

$$C_p(i,o) \stackrel{\text{def}}{=} i(x).\bar{o}\langle x\rangle.C_p\langle i,o\rangle$$

(Simplex means one-way communication.)

Leaking simplex channel:

$$C_l(i,o) \stackrel{\text{def}}{=} i(x).(\bar{o}\langle x\rangle + 0).C_l\langle i,o\rangle$$

Noisy simplex channel:

$$C_n(i,o) \stackrel{\text{def}}{=} i(x).(\bar{o}\langle x\rangle + \mathsf{new}\, y\, \bar{o}\langle y\rangle).C_n\langle i,o\rangle$$

Reordering simplex channel:

$$C_r(i,o) \stackrel{\text{def}}{=} i(x).(\bar{o}\langle x\rangle | C_r\langle i,o\rangle)$$

A channel C is said to be, for example, a reordering channel, if it is observation equivalent [25, Ch. 6&13] to, or weakly bisimilar to, C_r above,

$$C\langle i,o\rangle \approx C_r\langle i,o\rangle,$$

and so on for the other characteristic channels.

Note that one can make a channels with combined characteristics as follows:

$$C_{rl}(i,o) \stackrel{\text{def}}{=} \mathsf{new}\, o_r i_l\, (C_r\langle i,o_r\rangle | o_r(x).\bar{i_l}\langle x\rangle | C_l\langle i_l,o\rangle)$$

Channel C_{rl} will not be a (pure) reordering channel since it will not be observation equivalent to C_r. By an analogy to circuit design this could be seen as a 'serial composition' of two channels. Consider 'parallel composition' of channels, that is, sending the same data out into two different channels and then merging outputs from those two channels and it into a new output. The parallel combined channels does not make sense as channel: it may give more than one output for one input.

On the other hand, one can use a related idea to combine 2 simplex channels into a duplex channel, that is, a channel that permits 2-way communication between 2 parties, as is shown below.

Uniform duplex channel: Let $C(i,o)$ be a simplex channel. Then the following is a uniform duplex C-channel:

$$C_d(i_1,o_1,i_2,o_2) \stackrel{\text{def}}{=} C\langle i_1,o_1\rangle | C\langle i_2,o_2\rangle$$

A non-uniform duplex channel is made by combining two different simplex channels.

A limited capacity channel can be modeled in several ways depending on what to do with excess data, that is, data sent into a channel when the capacity is exceeded. Here are some alternatives for what the channel could do with such data:

- it is lost;
- the channel refuses to accept it.

4.3.2 Communications patterns

The trivial communication pattern involves two parties talking to each other. In this section we look at this and more advanced cases of communicating parties.

A broadcast channel mechanism, that is, a way of communicating messages to an unlimited number of recipients, seems difficult to implement in π-calculus. Note that one can make a one-time broadcast on a π-calculus channel b by using replication, but there is no way to stop the broadcast. This effectively renders the channel unusable. The distribution of fresh channels in which to perform broadcast leads to a bootstrap problem: new broadcast channels must be broadcast ...

Multi-cast communication can be achieved, though, but let us first look at the problem of interconnecting two actors, $A(i,o), B(i,o)$ that are both parametric in their input and output channels (be it π-calculus channels or other kinds of channels) as indicated by the argument names. The two actors can be connected together as follows:

$$\mathsf{new}\, a_i, a_o\, (A(a_i, a_o) | B(a_o, a_i))$$

This is a one-to-one setting where the two actors are equals.

Assume instead that an actor $A(i,o)$ wants to communicate with two other actors $B_1(i,o), B(i,o)$ in the following way:

- A's output should be input to both B_1, B_2;
- the output from both B_1, B_2 should be input to A;
- B_1, B_2 should not talk to each other.

Here is how to do it:

$$\mathsf{new}\, a_i a_o b_1 b_2\, (A(a_i, a_o) | !a_o(x).(\bar{b_1}\langle x\rangle | \bar{b_2}\langle x\rangle) | B_1(b_1, a_i) | B_2(b_2, a_i))$$

Here A could be seen as a server and B_1, B_2 as clients. Note that the B_i's must identify themselves to A if needed since A has no way of telling who it is receiving data form. This can be generalized to a one-to-many setting.

The listener pattern

Here we look at one particularly useful pattern of communication that we shall call the 'listener pattern' by analogy to the Unix network listener daemon.

Consider some process R that provides a resource for use by several other processes. There are two issues of concern in safe resource usage:

1. The integrity of resource R, that is, the use of it may require one-at-a-time access (using critical regions or similar mechanisms) in order to keep R's internal data structure consistent. This is an internal issue for R and we do not consider it further.

2. The integrity of communication with the resource. One must make sure that communication between an entity, A say, and R is not interfered with by another entity B. If A is to use R in non-trivial ways the two parties will need to engage in duplex communication, and then it is possible that messages from a third party on the same channel could be interpreted by A as coming from R or *vice-versa*.

For simplicity we assume that R has just a single channel r known to the outside world. The idea of the listener pattern of communication is that an actor A uses channel r as a means to obtain a fresh channel on which duplex communication between the two parties can take place undisturbed.

Assume that both A and R are parametric on the single channel where their duplex communication is to take place. Then the following new processes realizes safe resource usage:

$$L(l) \stackrel{\text{def}}{=} !\, l(c_A).\, \mathsf{new}\, r\, (\overline{c_A}\langle r\rangle | R\langle r\rangle)$$
$$A'(l) \stackrel{\text{def}}{=} \mathsf{new}\, a\, \bar{l}\langle a\rangle.a(c_R).A\langle c_R\rangle$$

A system of two actors using R through the listener process L is initiated as follows:

$$\mathsf{new}\, c\, (L\langle c\rangle | A'\langle c\rangle | A'\langle c\rangle)$$

All processes are parametric on a channel c that is used only initially, before the real resource usage starts. An actor (A' instance) sends a fresh channel a on the global channel (called l inside A') and this is read by L. Process L sends a fresh channel r back on the channel that it receives (called c_A inside L), and initializes the resource R with channel r. (Since we are not concerned with the internal integrity of R, we simply duplicate it for each new actor.) Process A' initializes A with the channel c_R that it received on channel a. Finally, the processes R and A are can communicate safely.

4.4 Modeling sessions

In chapter 4 we mentioned several examples of 'sessions'. We shall describe one typical usage of a session, and present a model for it in terms of a π-calculus specification. As indicated by our simple example, several other specifications of 'session' should be given, but we leave it for future work.

4.4.1 A session on a remote host machine

The context of usage is the following: Assume that we have a host machine H, and that you are a guest G. The host H permits one or several guests $G_1, \ldots, G_n$ to use a set of resources R on the host machine H. This is done in two ways. First, H can be accessed from a remote machine G, by knowing a code (username and password) to login to H. In other words, we presuppose that the knowledge of G's name is known to the community of machines able to be involved in an reaction.

Then G might run several services on H and log-off at the end of the day.

Key concepts in this session:

- reserved (unique) channel
- encapsulation
- login/logoff
- prevail extended resources

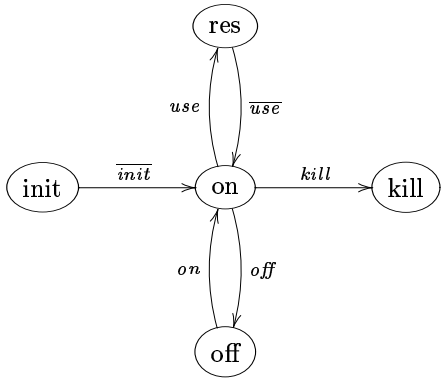


Figure 4.7: Session commands and states.

- Partitioning of H / Protection of G's

Encapsulation can be implemented in several ways. One way is to hide processes by a restriction.

4.4.2 A ghost-session on a remote host machine

Similar to the context above, but in addition, it should be possible for G to logg off at the end of the day where H preserves the processes run by G. The life cycle of a session is characterized by the phases:

1. The guest G logg in to the host H - the opening of a session
2. The guest G logg off the host H - but is keeping the session alive
3. G being logged on, and using the recourses H is providing
4. The guest G is logging off the host H.

- identity of user
- protection of G's processes from attack outside
- freeze/encapsulation of processes

4.4.3 Simple session modeled in π-calculus

This section illustrates how to model a simple, yet meaningful session concept in π-calculus. The situation we want to model is the following:

There exists a resource *Res* and a number of actors. The actors' access to the resource can only happen a controlled manner using 'sessions'. A session is obtained from a listener process in a way similar to Section 4.3.2. The session process only understands a few 'commands', sends out a few 'commands', and can be in one of five states, see the finite-state machine in Figure 4.7.The session regulates access to R. Using the command channel *use* the session mediates duplex communication between an actor A and the resource *Res*. The semantics

of *off* is that A leaves the session to (possibly) compute on its own, while *on* is used to re-active the session. The actor uses *kill* to kill the session. Note in particular that A cannot use the resource (command *use*) when the session is in state 'off'—any attempt to communicate on channel *use* will be delayed by the session.

Here are the listener and session processes that realizes the above:

$$Listener(l) \stackrel{\text{def}}{=} !\, l(c).\,\mathsf{new}\, init\, (\bar{c}\langle init\rangle |\, \mathsf{new}\, res\, (Res\langle res\rangle\ | Session\langle init, res\rangle))$$

$$Session(init, res) \stackrel{\text{def}}{=} (\mathsf{new}\, on\ off\ use\ kill\,)\ \overline{init}\langle on, off, use, kill\rangle.Session'\langle on, off, use, kill, res\rangle$$

$$\begin{aligned} Session'(on, off, use, kill, res) \stackrel{\text{def}}{=}\ & off.on.Session'\langle on, off, use, kill, res\rangle \\ & + use(x).\overline{res}\langle x\rangle.res(y).\overline{use}\langle y\rangle.Session'\langle on, off, use, kill, res\rangle \\ & + kill \end{aligned}$$

4.5 Tools

4.5.1 Buffer

A buffer is a general mechanism used to store messages in order to accomplish asynchronous communication. One operation should be used to create a buffer, and three external operation on it: insert a message, remove a message and kill the buffer. As long as the buffer exists, assume there are three channels associated with it, for the time being called: p, c and m, denoting *producer*, *consumer* and *management*. Call the buffer process B, and we expect that a producer process may interact with it as:

$$\bar{p}\langle v\rangle.P|B \longrightarrow P|B'$$

where v is a channel representing some value. Similarly with consumers:

$$c(x).Q|B \longrightarrow \{v/x\}Q|B'$$

assuming v is a channel representing the first value in the buffer B.

Internals

An operating buffer cell that is neither the first nor the last in the buffer might be represented as follows:

$$\mathrm{Cell}(t, v, n) = t(x).\bar{x}\langle v\rangle.\bar{n}\langle x\rangle$$

The channel t symbolizes that the cell is waiting for the token, x which tells it is the first in queue. This will be the same as the channel c for consumers. Whenever it receives this, it is the first in the queue, and anyone "listening" on

this channel will receive this cell's value, namely v. At last, the cell must pass over its token to the next cell, n and then — *die*.

Producers need to access the last cell in order to (1) generate a new cell, and (2) tell the current last cell where to pass over its token when time due. Like this:

$$\text{Last}(t, v, p) = \mathsf{new}\, n\, (\text{Cell}\langle t, v, n\rangle | p(y).\text{Last}\langle n, y, p\rangle)$$

So that whenever a cell is created, it becomes the current last cell, and a process is ready to accept a new message from a producer and make still another last cell, and so on. Finally, we need a system to start the whole process:

$$\text{Buffer}(p, c) = \mathsf{new}\, m\ (\bar{m}\langle c\rangle \mid p(y).\text{Last}\langle m, y, p\rangle)$$

Two reasons for this construction: First, it waits for the first message to be received ($p(y)$)and then activates a cell (last and $\bar{m}\langle c\rangle$) in order to store it and pass the token to the next last process. Second, whenever the buffer is empty (produced messages is equal to consumed) the process expression achieved will be structural congruent to the above definition.

To test the buffer, lets simulate two producers and two consumers and write out the reaction chain:

$$(\bar{p}\langle v\rangle.\bar{p}\langle v'\rangle.c(x).c(y).\text{Use}\langle x, y\rangle) \mid \text{Buffer}(p, c)$$

This will lead to the reaction chain show in Figure 4.8.

To create a buffer, one just issues the following process:

$$\text{BufferCreate} = (b)\ ![b(r).\, \mathsf{new}\, p, c\, (\bar{r}\langle pc\rangle | \text{Buffer}\langle p, c\rangle)]$$

Datatypes

Datatypes in π-calculus are abstract processes. A particular value is also an abstract process, and the binding of a value to a variable is represented as an input channel and a process.

Consider an enumeration example as in Milner's book [25]. In algebraic specification, one would write this out as:

$$\text{Male}: \longrightarrow \mathbf{Person} \tag{4.1}$$
$$\text{Female}: \longrightarrow \mathbf{Person} \tag{4.2}$$
$$\text{Child}: \longrightarrow \mathbf{Person} \tag{4.3}$$

These functions, Male, Female and Child, are called *generators*. When functions are made, one *case split* on the generators. Like:

$$\text{CanBeFather}: \mathbf{Person} \longrightarrow \mathbf{Boolean} \tag{4.4}$$
$$\text{CanBeFather(Male)} = \text{true} \tag{4.5}$$
$$\text{CanBeFather(Female)} = \text{false} \tag{4.6}$$
$$\text{CanBeFather(Child)} = \text{false} \tag{4.7}$$

In π-calculus, one represent the generators as channels, and a datatype value returns the channel it "identifies" itself with:

$$
\begin{aligned}
&[(\bar{p}\langle v\rangle.\bar{p}\langle v'\rangle.c(x).c(y).\mathrm{Use}\langle x,y\rangle)] \mid \underline{\mathrm{Buffer}(p,c)} = \\
&\left[\underline{\bar{p}\langle v\rangle}.\bar{p}\langle v'\rangle.c(x).c(y).\mathrm{Use}\langle x,y\rangle\right] \mid \mathsf{new}\, m\ \left(\bar{m}\langle c\rangle|\underline{p(y)}.\mathrm{Last}\langle m,y,p\rangle)\right) \longrightarrow \\
&[\bar{p}\langle v'\rangle.c(x).c(y).\mathrm{Use}\langle x,y\rangle] \mid \mathsf{new}\, m\ \left(\bar{m}\langle c\rangle|\underline{\{v/y\}}\mathrm{Last}\langle m,y,p\rangle\right) = \\
&[\bar{p}\langle v'\rangle.c(x).c(y).\mathrm{Use}\langle x,y\rangle)] \mid \mathsf{new}\, m\ \left(\bar{m}\langle c\rangle|\underline{\mathrm{Last}\langle m,v,p\rangle}\right) = \\
&[\bar{p}\langle v'\rangle.c(x).c(y).\mathrm{Use}\langle x,y\rangle)] \mid \mathsf{new}\, m\ \left[\bar{m}\langle c\rangle|\,\mathsf{new}\, n\ \left(\underline{\mathrm{Cell}\langle m,v,n\rangle}|p(y).\mathrm{Last}\langle n,y,p\rangle\right)\right] = \\
&[\bar{p}\langle v'\rangle.c(x).c(y).\mathrm{Use}\langle x,y\rangle] \mid \mathsf{new}\, m\ \left[\underline{\bar{m}\langle c\rangle}|\,\mathsf{new}\, n\ \left(\underline{m(x)}.\bar{x}\langle v\rangle.\bar{n}\langle x\rangle \mid p(y).\mathrm{Last}\langle n,y,p\rangle\right)\right] \longrightarrow \\
&[\bar{p}\langle v'\rangle.c(x).c(y).\mathrm{Use}\langle x,y\rangle] \mid \mathsf{new}\, m\ \left[\underline{0}\mid\,\mathsf{new}\, n\ \left(\underline{\{c/x\}}\bar{x}\langle v\rangle.\bar{n}\langle x\rangle \mid p(y).\mathrm{Last}\langle n,y,p\rangle\right)\right] \equiv \\
&\left[\underline{\bar{p}\langle v'\rangle}.c(x).c(y).\mathrm{Use}\langle x,y\rangle\right] \mid \mathsf{new}\, m\ \left[\mathsf{new}\, n\ \left(\bar{c}\langle v\rangle.\bar{n}\langle c\rangle \mid \underline{p(y)}.\mathrm{Last}\langle n,y,p\rangle\right)\right] \longrightarrow \\
&[c(x).c(y).\mathrm{Use}\langle x,y\rangle] \mid \mathsf{new}\, mn\ \left[\bar{c}\langle v\rangle.\bar{n}\langle c\rangle \mid \underline{\{v'/y\}}\,\mathrm{Last}\langle n,y,p\rangle\right] \equiv \\
&\left[\underline{c(x)}.c(y).\mathrm{Use}\langle x,y\rangle\right] \mid \mathsf{new}\, mn\ \left[\underline{\bar{c}\langle v\rangle}.\bar{n}\langle c\rangle \mid \mathrm{Last}\langle n,v',p\rangle\right] \longrightarrow \\
&\left[\underline{\{v/x\}}.c(y).\mathrm{Use}\langle x,y\rangle\right] \mid \mathsf{new}\, mn\ [\bar{n}\langle c\rangle \mid \mathrm{Last}\langle n,v',p\rangle] \longrightarrow \\
&[c(y).\mathrm{Use}\langle v,y\rangle] \mid \mathsf{new}\, mn\ \left[\bar{n}\langle c\rangle \mid \underline{\mathrm{Last}\langle n,v',p\rangle}\right] = \\
&[c(y).\mathrm{Use}\langle v,y\rangle] \mid \mathsf{new}\, mn\ [\bar{n}\langle c\rangle \mid \underline{\mathsf{new}\, n'}\ (\mathrm{Cell}\langle n,v',n'\rangle \mid p(y).\mathrm{Last}\langle n',y,p\rangle)] \equiv \\
&[c(y).\mathrm{Use}\langle v,y\rangle] \mid \mathsf{new}\, mnn'\ \left[\bar{n}\langle c\rangle \mid \underline{\mathrm{Cell}\langle n,v',n'\rangle} \mid (p(y).\mathrm{Last}\langle n',y,p\rangle)\right] = \\
&[c(y).\mathrm{Use}\langle v,y\rangle] \mid \mathsf{new}\, mnn'\ \left[\underline{\bar{n}\langle c\rangle} \mid \left(\underline{n(x)}.\bar{x}\langle v'\rangle.\bar{n'}\langle x\rangle\right) \mid (p(y).\mathrm{Last}\langle n',y,p\rangle)\right] \longrightarrow \\
&[c(y).\mathrm{Use}\langle v,y\rangle] \mid \mathsf{new}\, mnn'\ \left[\underline{0}\mid\ \left(\underline{\{c/x\}}\bar{x}\langle v'\rangle.\bar{n'}\langle x\rangle\right) \mid (p(y).\mathrm{Last}\langle n',y,p\rangle)\right] \equiv \\
&\left[\underline{c(y)}.\mathrm{Use}\langle v,y\rangle\right] \mid \mathsf{new}\, mnn'\ \left[\left(\underline{\bar{c}\langle v'\rangle}.\bar{n'}\langle c\rangle\right) \mid (p(y).\mathrm{Last}\langle n',y,p\rangle)\right] \equiv \\
&\left[\underline{\{v'/y\}}.\mathrm{Use}\langle v,y\rangle\right] \mid \mathsf{new}\, \underline{mn}n'\ \left[\left(\bar{n'}\langle c\rangle\right) \mid (p(y).\mathrm{Last}\langle n',y,p\rangle)\right] \equiv \\
&[\mathrm{Use}\langle v,v'\rangle] \mid \underline{\mathsf{new}\, n'\ \left[\left(\bar{n'}\langle c\rangle\right) \mid (p(y).\mathrm{Last}\langle n',y,p\rangle)\right]} = \\
&[\mathrm{Use}\langle v,v'\rangle] \mid [\mathrm{Buffer}\langle p,c\rangle]
\end{aligned}
$$

Figure 4.8: Two producers, two consumers and a buffer.

$$\text{Man} = (x)x(mwc).\bar{m}\langle\rangle \tag{4.8}$$
$$\text{Woman} = (x)x(mwc).\bar{w}\langle\rangle \tag{4.9}$$
$$\text{Child} = (x)x(mwc).\bar{c}\langle\rangle \tag{4.10}$$

Case splitting in π-calculus is implemented with three processes:

$$\text{CanBeFather} = (x, r)\,\mathsf{new}\, m, w, c\,\bar{x}\langle mwc\rangle.(m().\text{true}\langle r\rangle|w().\text{false}\langle r\rangle|c().\text{false}\langle r\rangle)$$

This process "asks" whether the channel x represents a man, woman or a child, and returns true or false binded to r.

Natural numbers

A little bit more complicated example is to represent natural numbers. The generators are:

$$0\colon \longrightarrow \mathbf{Nat} \tag{4.11}$$
$$S\colon \mathbf{Nat} \longrightarrow \mathbf{Nat} \tag{4.12}$$

And we can define plus like

$$+\colon \mathbf{Nat} \times \mathbf{Nat} \longrightarrow \mathbf{Nat} \tag{4.13}$$
$$x + 0 = x \tag{4.14}$$
$$x + S\,y = S(x + y) \tag{4.15}$$

In π-calculus, we get the following:

$$0 = (c)c(ns).\bar{n}\langle\rangle \tag{4.16}$$
$$S(x) = (c)c(ns).\bar{s}\langle x\rangle \tag{4.17}$$
$$\text{Plus}(x, y) = (r)\,\mathsf{new}\, n, s\,\bar{y}\langle ns\rangle.\quad n().\text{copy}\langle x, r\rangle\ | \tag{4.18}$$
$$s(y').r(ns).\,\mathsf{new}\, r'\ \bar{s}\langle r'\rangle\text{Plus}\langle x, y'\rangle\langle r'\rangle \tag{4.19}$$

References

1. M. Abadi, A. D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Computation and Information*, 2000. to appear.

2. G. Boolos. *Logic, Logic and Logic.* Harvard University Press, 2000.

3. P. J. Braam. The coda distributed file system. *Linux Journal*, (50), June, 1998.

4. L. Cardelli, A. D. Gordon. Mobile ambients. In *FoSSaCS'98: Foundations of Software Science and Computation Structure, First International Conference*, no. 1378 in Lecture Notes in Computer Science, pp. 140–155. Springer-Verlag, 1998. Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98.

5. L. Cardelli, A. D. Gordon. Types for mobile ambients. In A. Aiken (ed.), *POPL '99: The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 79–92. ACM Press, 1999.

6. L. Cardelli, A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In T. Reps (ed.), *POPL '00: The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 365–377. ACM Press, 2000.

7. L. Cardelli, A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

8. A. J. Demers, *et al.* The Bayou architecture: Support for data sharing among mobile users. In *Proc. of the Workshop on Mobile Computing Systems and Applications*, December, 1994.

9. G. Frege. *Begriffschrift: Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Halle, 1879.

10. G. Frege. *Foundations of Arithmetics.* Oxford University Press, 1884. Translation from Breslau: Wilhelm Koebner.

11. G. Frege. Function and concept. In *Translations from the Philosophical Writings of Gottlob Frege.* Oxford University Press, 1891.

12. G. Frege. On concept and object. In *Translations from the Philosophical Writings of Gottlob Frege.* Oxford University Press, 1892.

13. G. Frege. On sense and meaning. In *Translations from the Philosophical Writings of Gottlob Frege*. Oxford University Press, 1892.

14. G. Frege. *Grundgesetze der Arithmetic*. Georg Oms, 1893.

15. G. Frege. Thoughts. In *Logical Investigations*. Basil Blackwell, 1918.

16. R. Goldblatt. *Logics of Time and Computation*. CSLI, 1992.

17. J. Hintikka. *Knowledge and Beliefs: an introduction to the logic of the two notions*. Ithaca N.Y., Cornell University Press, 1962.

18. ISO. Open distributed processing - reference model - part 1 overview, 1995. Obtained from the Open Group.

19. ISO. Open distributed processing - reference model - part 2 foundations, 1995. Obtained from the Open Group.

20. J. Jing, A. Helal, A. Elmagarmid. Client-server computing in mobile environments. *ACM Computing Surveys*, 31(2):117–157, 1999.

21. A. D. Joseph, *et al.* Rover: A toolkit for mobile information access. In *Proc. of the fifteenth Symposium on Operating Systems Principles*, December, 1995.

22. A. D. Joseph, K. M. Frans. Building reliable mobile-aware applications using the rover toolkit. In *Proc. of the 2nd ACM Conference on Mobile Computing and Networking*, November, 1996.

23. A. D. Joseph, J. A. Tauber, K. M. Frans. Mobile computing with the rover tookit. *IEEE Transactions on Computers*, February 1997.

24. D. K. Lewis. *Counterfactuals*. Harvard University Press, 1973.

25. R. Milner. *Communicating and Mobile Systems: the pi-Calculus*. Cambridge University Press, 1999.

26. R. Milner, J. Parrow, D. Walker. A calculus of mobil processes part I and II. *Information and Computation*, 100(1):1–77, 1992.

27. R. Milner, J. Parrow, D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

28. J. C. Mitchell. *Foundations for Programming Languages*. Foundations of Computing Series. The MIT Press, 1996.

29. A. Moen. The concept 'session' modelled from below. In *The 12th Nordic Workshop on Programming Theory (NWPT'00), Bergen, October 11-13*, 2000.

30. B. Noble, M. Satyanarayanan. Experience with adaptive mobile applications in odyssey. *Mobile Networks and Applications*, 4, 1999.

31. B. D. Noble. System support for mobile, adaptive applications. *IEEE Personal Communications*, 7(1):44–49, February 2000.

32. J. Parrow. An introduction to the π-calculus. Chapter to appear in *Handbook of Process Algebra*, ed. Bergstra, Ponse and Smolka, Elsevier.

33. S. L. Peyton Jones. *The implementation of functional programming languages.* International series in computer science. Prentice Hall, 1987.

34. M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1), February, 1996.

35. M. Satyanarayanan. Coda: A highly available file system for a distributed workstation environment. In *Proc. of the 2nd IEEE Workshop on Operating Systems*, September, 1989.

36. K. Segerberg. *Dynamic Logic.* unpublished, 1993.

37. A. S. Tanenbaum. *Distributed Operating Systems.* Prentice Hall International Editors, 1995.

38. A. S. Tanenbaum. *Computer Networks.* Prentice Hall, 1996.

39. D. B. Terry, *et al.* Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proc. of the 15th Symposium on Operating Systems Principles (SOSP-15)*, December, 1995.

40. D. B. Terry, *et al.* Session guarantees for weakly consistent replicated data. In *Proc. of the International Conference on Parallel and Distributed Information Systems (PDIS)*, September, 1994.

41. M. Theimer, *et al.* Dealing with tentative data values in disconnected work groups. In *Proc. of the Workshop on Mobile Computing Systems and Applications*, December, 1994.

42. G. H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.