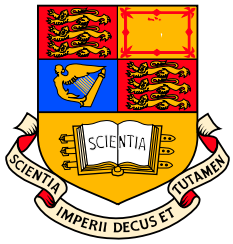
University of London
Imperial College of Science, Technology and Medicine
Department of Computing



Towards Abstractions For Distributed Systems

Martin Berger

A thesis submitted for the degree of Doctor of Philosophy
of the University of London
and for the Diploma of Imperial College.

Revised Version (of November 24, 2004)

"Não sou nada.
Nunca serei nada.
Não posso querer ser nada.
À parte isso, tenho em mim todos os sonhos do mundo."

Abstract

For historical, sociological and technical reasons, λ-calculi have been the dominant theoretical paradigm in the study of functional computation. Similarly, but to a lesser degree, π-calculi dominate advanced mathematical accounts of concurrency. Alas, and despite its ever increasing ubiquity, an equally convincing formal foundation for distributed computing has not been forthcoming. This thesis seeks to contribute towards ameliorating that omission. To this end, guided by the assumption that distributed computing is concurrent computing with partial failures of various kinds, we extend the asynchronous π-calculus with

- a notion of sites,
- the possibility of site failure,
- a persistence mechanism to deal with site failures,
- the distinction between inter-site and intra-site communication,
- the possibility of message loss in inter-site communication and
- a timer construct, as is often used in distributed algorithms to deal with various failure scenarios.

The basic theory of two of the resulting augmented π-calculi is explored in considerable detail: the asynchronous π-calculus with timers and the asynchronous π-calculus with timers, sites and message failure. The emphasis of this development is on soundly approximating reduction congruence, the canonical equivalence for asynchronous π-calculi. In the case of the asynchronous π-calculus with timers we manage to obtain a characterisation of reduction congruence as a labelled bisimilarity. Our results appear to be robust under variations of the underlying calculi. In addition, as a test to evaluate the usefulness and the integration of the aforementioned extensions, we cleanly and incrementally represent the Two Phase Commit Protocol – an important distributed algorithm, used to ensure consistency of distributed databases in the face of various kinds of (non-byzantine) failures – in the asynchronous π-calculus, the asynchronous π-calculus with timers, sites and message failure and in the asynchronous π-calculus with timers, sites, site failure and

message failure. We establish the correctness of our representations by equational reasoning.

Attribution

The results presented in this thesis are all by the author, except for the encodings of the Two Phase Commit Protocol in Chapters 3, 6 and 8, which were obtained in collaboration with Kohei Honda, as reported in [13, 14]. The corresponding correctness proofs for this protocol were inspired by this collaboration, too, but differ substantially in the details. Kohei Honda also suggested locality as a suitable restriction for processes executing in sites, cf. Chapter 5.

Contents

List of Figures

Notation and Mathematical Preliminaries

We assume the usual set-theoretic foundations of mathematics, in particular ordinals and well-founded induction. We use $\subseteq$ to denote subset inclusion and $\subset$ but also $\subsetneq$ for strict inclusion. We write $\mathcal{P}(A)$ for the *powerset* of A and $\mathcal{P}_{fin}(A)$ for the set of all finite subsets of A. We write $\vec{x}$ to denote tuples $\langle x_0 ... x_{n-1}\rangle$ $(n \geq 0)$ and $\{\vec{x}\}$ is a shorthand for $\{x_0 ... x_{n-1}\}$, ie. duplicates will be removed. The *natural numbers* are the set $\mathbb{N} = \{0, 1, 2, 3, 4, ...\}$, while *integers* are $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, 3, 4, ...\}$. We denote the *reals* by $\mathbb{R}$, the *non-negative reals* by $\mathbb{R}^+$, the *rationals* by $\mathbb{Q}$ and the non-negative rationals by $\mathbb{Q}^+$. If $f : A \to B$ is a function, we write $\mathsf{dom}(f)$ for A, the *domain* of f, and $\mathsf{cod}(f)$ for the *codomain* B. A *partial function* f from A to B is denoted $f : A \rightharpoonup B$. The *universal relation* on a set A is $A \times A$. If $R \subseteq A \times B$ is a binary relation, then its *transposition* is $R^{-1} = \{(b, a) \mid (a, b) \in R\}$. The *transitive closure* of R is R^*. Some binary relations will also be referred to by an arrow like $\to$, in which case the transitive, reflexive closure is $\twoheadrightarrow$. If $R \subseteq A \times B$ and $S \subseteq B \times C$ are relations, then we often abbreviate their relational composition $S \circ R \subseteq A \times C$ to RS. Please note the reversal! If $\to$ is a binary relation, $\to_n$ is a shorthand for $\underbrace{\to \cdots \to}_{n}$. We write $\to^+$ if $\to_n$ for some $n > 0$. If $\sqsubseteq$ is a binary relation on A, $a \in A$ is *maximal* if whenever $b \in A$ and $a \sqsubseteq b$, then $a = b$. If $b \sqsubseteq a$ for all $b \in A$ then a is A's *maximum* element. We will often use ternary relations $\to \subseteq A \times B \times A$ and we write $a \xrightarrow{b} a'$ instead of $(a, b, a') \in \to$. If $a_0 \xrightarrow{b_0} a_1 \xrightarrow{b_1} \cdots a_{n-1} \xrightarrow{b_{n-1}} a_n$ and the details of $a_1, ..., a_{n-1}$ don't matter, we may write $a_0 \xrightarrow{\langle b_0,...,b_{n-1}\rangle} a_n$. A *graph* is a pair $(V, \to)$ where V is the *carrier set* of the graph and $\to$ is its *reduction relation*. A *string* over a set A is an n-tuple of A's elements, where $n \in \mathbb{N}$. We usually write $\langle a_0, ..., a_{n-1}\rangle$ for a string of *length* n: $\mathsf{length}(\langle a_0, ..., a_{n-1}\rangle) = n$. The *empty string* is written $\langle\rangle$. A *signature* Σ is a tuple $(S_1, \cdots, S_m, f_1, \cdots, f_n, \#)$. Each S_i is a *sort-symbol* and the f_i are *function-symbols*. The *arity-function* $\#$ maps f_i to a tuple $\#(f_i) = (S_{i_1}, \cdots, S_{i_{k_i}}, S)$ of sort-symbols. Given Σ as above, a Σ-*algebra* $\mathfrak{A}$ is a tuple $\mathfrak{A} = (S_1^{\mathfrak{A}}, \cdots, S_m^{\mathfrak{A}}, f_1^{\mathfrak{A}}, \cdots, f_n^{\mathfrak{A}})$ where each $S_i^{\mathfrak{A}}$ is a set and $f_i^{\mathfrak{A}}$ is a function such that $\#(f_i) = (S_{i_1}, \cdots S_{i_{k_i}}, S)$ implies $f_i^{\mathfrak{A}} : S_{i_1}^{\mathfrak{A}} \times \cdots \times S_{i_{k_i}}^{\mathfrak{A}} \to S^{\mathfrak{A}}$. We assume

the usual notions of *term algebra* and *polynomial* over Σ.

Chapter 1

Introduction

This chapter introduces the problem we are trying to solve and proposes answers to two questions: what are distributed systems? And: why use the π-calculus as a basis for modelling them?

1.1 The Problem

Distributed computing's pervasiveness, already considerable at the time of writing, will continue to grow dramatically as a consequence of the rapid computerisation of industrial society. Examples of currently operating world-wide distributed computing systems are the Internet, the telephone network and the information infrastructures of globally acting organisations such as military services and multinational companies. The evolution of computer mediated service provision, often referred to under the simplifying name "electronic commerce", will increase the dependency of a significant part of the world's population on distributed computing systems. This raises the question of their reliability.

Currently, the situation seems rather bleak. Essentially, there are two established ways of exploring the reliability of computing systems: testing and correctness proofs. Although it can be argued that this distinction is a spurious one since proofs are best understood as non-standard tests – as they are valid only under ultimately unverified correctness assumptions, for example pertaining to the ability of proof generating and verifying agents, whether human or automated, to avoid invalid deductions – the distinction has much pragmatic appeal. Both, testing and correctness proofs are often not easy to get right, even in a sequential setting. Introduction of concurrency, which properly contains sequential computation, and distribution, itself encompassing concurrent computation, makes matters worse. Non-determinism, interference, deadlocks, starvation, unfairness, message loss, partial process failure and recovery in addition to a host of other phenomena greatly complicate testing and generation of correctness proofs. Consequently, testing or correctness proofs of

distributed systems are usually partial, giving little reason to be unconcerned.

These difficulties non-withstanding, many distributed computing systems are often considered to work reasonably or even remarkably well. Whether this is due to the ingenuity of their designers or to low user expectations is an interesting question, but the ever-increasing ubiquity of such systems suggests that they are indeed reliable enough for many applications. Nevertheless, for at least the following reasons it would seem appropriate to strive for improving the state-of-the-art.

- More and more, human well-being depends on properly functioning of distributed systems.
- Increasing complexity and interdependence of distributed systems might make some form of failure more likely or more disastrous.
- Understanding complex systems and discovering subtle bugs is aesthetically pleasing.

This text seeks to help providing some groundwork towards more reliable distributed systems. More specifically, it seeks to alleviate one of the shortcomings of current methodologies for reasoning about distributed systems, the lack of convenient and rigorous models of distributed computations, by providing fully formal models of the following features of concurrent systems in a unified, process calculus framework: time, location, location failure, persistence and non-byzantine message-failure.

1.2 Distributed Systems?!?

But what are Distributed Systems (or DS for short)? It would be nice if we could begin with an incontrovertible definition of what will be our subject matter for the next 230 or so pages. Alas this cannot be done in good conscience, because the definition is part of the problem ... More precisely, nailing down the essence of the phenomena that we associate with the fuzzy designator DS is one of the unsolved problems characterising this field. The degree of confusion can be gauged by comparing the respective first pages of two popular DS textbooks. "*A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages*" is how Coulouris, Dollimore and Kindberg begin their [30], while Nancy Lynch claims, on the first proper page of her monumental treatise [70] on distributed algorithms, that "*[d]istributed algorithms run on a collection of processors, which need to communicate somehow. Some common methods of communication include accessing shared memory, sending point-to-point or broadcast messages (either over a long distance or local area network), or executing remote procedure calls*". How is a little guy[1] to avoid confusion

[1] i.e. the present author, we, I, Martin Berger or maybe even "the present author".

if even the big guns seem unable to? Fortunately, the process calculus perspective on computing, with its mantra "computation is interaction" affords a way out of this conundrum by representing interactions via shared memory as a peculiar form of message passing. Anyway, coming back to characterising DS, we may arrive at a sufficient understanding of DS by an idealising account of the historical development of the usage of the term.

Once upon a time, when the world was still decent, computers where *centralised*. Despite being complex, chaotic and analog physical devices (like everything else), their organisation and the environments they operated in allowed them to exhibit *emergent* or *macroscopic properties* [109] such as being digital, acting as if they were finite state automata or Turing Machines, sequentially executing a program. Another emergent property was that of being error free, or, if not error free, then at least failing so rarely that few cared about the possibility. Or, most relevantly here, if they'd fail, they'd do so cleanly or *totally*. That means they'd completely and suddenly stop to function, without possibility of any further interaction or observation of computation. This totality of failure of centralised computation is their key difference from DS, but it should not be forgotten that it is "just" an observer construct in the sense that if we'd bother to look at enough details, maybe at the level of individual transistors, what appeared a total failure to the casual observer became partial. It just happens that our interactions/observations with centralised computation are mostly such that we don't care about this, or at least don't look hard enough to notice. Of course this is always the case with emergent properties: the reason that emergent, i.e. inaccurate description works, in the sense of allowing to make predictions that are not falsified by (subsequent) observations, is simply limited observational capabilities.

As computers became more popular, they were connected, for example to share expensive resources such as printers, or to pool their processing power for solving some computationally expensive problem. The connection mechanism is now mostly called "the net", but *"benign(ish) wires"* may be a more descriptive phrase because it captures the behavioural results of the (usually considerable) efforts the interconnection has to invest to be of use. A benign wire would ideally transmit data entered at one end to the other, without any delay or modification. The qualification "'ish" is intended to convey that this cannot always be achieved: messages get lost, duplicated and even corrupted, although corruption is much less frequent than loss or duplication. In addition, messages are subject to often unpredictable transmission delays, significantly exceeding the duration of atomic steps in the computations of the connected computers. These imperfections in the interconnecting benign(ish) wires have serious ramifications for the programs executed on the individual computers. They are the reason for distinguishing DS from other types of computation.

- Messages sent to other computers can no longer be assumed to arrive. More generally, the points in time when messages will arrive (possibly more or less than one, due to message loss or duplication) cannot be predicted by the sender. This phenomenon is the *asynchrony* of DS.
- In addition, DS *fail partially*: an observer may interact with one networked computer, while another has crashed. The failed computer still fails totally, but the computation it carried out was but a part of a larger computational process, other fragments of which can still be observed.

Both phenomena lead to what may be called *informational uncertainty*, the inability of the participating computations to have precise information about the overall state of the entire system. Information's accuracy seems negatively correlated with its distance from whatever it describes.

1.3 Why Use π-Calculi as a Basis for Modelling Distributed Systems?

> "I make a disclaimer. I reject the idea that there can be a unique conceptual model, or one preferred formalism, for all aspects of something as large as concurrent computation, which is in a sense the *whole* of our subject – containing sequential computation as a well-behaved special area." Robin Milner [76]

Asynchrony is not germane to DS. As just one example, the asynchronous π-calculus, the formal basis of our subsequent development, was explicitly designed to be a model of concurrent computation where message senders cannot be sure about when their messages are received. In addition, all process calculi can be considered to model partial computation because they allow more than one process to execute in parallel. One may argue that this is not partiality in the sense of DS, because there is no notion of failure. But clearly, one key feature of partial failure is present: a spatial partition of the computation. It seems that the demarcation between concurrent and distributed computations is blurred, justifying the opening gambit of claiming the definition of DS was part of the problem under investigation.

Anyway, this proximity is a good thing! As π-calculi have proven to be powerful tools for modelling computation, we may already have a well-developed formal basis upon which to build a comprehensive mathematical theory of DS. The hope is that a few well-chosen extensions to a powerful process calculus are sufficient for capturing all key elements of DS. That is the guiding assumption behind this thesis. It may be in blatant contradiction with the Milner quote above, but without trying one cannot work out who's correct. Less obvious than the π-calculus as a starting point is what these extensions should be. Our proposals centre around providing a clearcut unit of failure, a point of reference with respect to which various failures and means for their neutralisation can be formalised.

Once this approach is accepted, the details of a suitable calculus to be extended need to be identified. There are many to choose from. We have opted for the *asynchronous* π-calculus for three reasons. Firstly, we are rather familiar with it. Secondly because it "feels the right choice" and thirdly because a decade of research on programming language expressivity has firmly put the asynchronous π-calculus in the lead when it comes to convenient encodings of other computational formalisms. This is evidence, though not compelling, that asynchronous name passing interaction is indeed a powerful tool for semantic investigations.

Once extensions have been found and integrated, we need to ask if the result is any good. It may be too early to say, not only because currently there are rather few alternative proposals to compare against, but also because the quality of a formalism lives mostly in the theoretical and practical developments it inspires. Once they roll in, we hope they'll show that our proposals exhibit the following positive properties.

- Clarity, simplicity, elegance.
- Rigour, which means that everything relevant can be expressed in the language of modern mathematics.
- We want to compare and integrate. This requires a certain uniformity of the formalism and harmonious coexistence not only with the other extensions, but also with the underlying π-calculus.
- Minimality, orthogonality. Each fundamental phenomenon of distributed systems should be modelled cleanly by one and only one construct. Each proposed extension should be indispensable in the sense that in its absence, an essential feature could not be expressed very well.
- The extensions should be incremental. This means that the existing π-calculus technology and results should not be invalidated by the extensions. On the contrary, it would be good if they could be recovered as canonical special cases. In particular, we would like to retain reduction semantics, name-passing synchronisation trees and bisimulations as key reasoning tools.
- Teachability. Our models for distributed systems should not only useful for theoretical development but also for educational purposes.

1.4 Our Contributions

Starting from the asynchronous π-calculus, we propose three simple extensions.

- Timers and a notion of discrete time. This is the most drastic extension, because it changes the computational behaviour more than our other modifications. Nonetheless, timers are already well established extensions to process calculi, so the syntactic details of this proposal should be uncontroversial. Our contribution here is being the first to investigate the effect of timers on π-calculi in earnest. This means an in-depth study of some of the canonical equivalences, with the highlight being a characterisation of reduction congruence as a labelled bisimulation. Comparable characterisations for other π-calculi have so far proved elusive. That complicating a calculus seems to simplify equational reasoning tools is a curious and as yet ill-understood phenomenon.

- Sites and message failure. Sites, also called locations, are the point for reference for failures. Explicitly representing sites in syntax appears necessary, because processes in the π-calculus have a somewhat under-determined syntax in the sense that the structural congruence $\equiv$ allows to modify syntactic details to a considerable extent without semantic effect. This prevents the division of processes into autonomous parts that could serve as a stable basis for introducing the possibility of failures in communication without additional syntax. We propose a simple syntactic addition and study the resulting calculus. In particular, we extract a particular form of timer usage that can serve as a basis for a theory of converting from non-distributed into distributed algorithms in a semantics-preserving way.

- Site failures and savepointing to model persistence. Our treatment of this extension is much less thorough than that of the previous two, because we are much less sure about its appropriateness. All we do is to propose the mechanism and test it in an example.

Have our extensions met our stated criteria above? These criteria are mostly aesthetic in nature and hard to apply as objective evaluation criteria. We leave it for the readers to decide ...

References to future aesthetic judgements may be seen as a bit of a cop-out, so this thesis does something else, an "empirical" experiment: we take a fundamental distributed algorithm, the Two Phase Commit Protocol, and try to express it in the proposed calculi. If that were impossible, we surely had failed. Fortunately, this turns out not to be the case. The algorithm scales gracefully as we turn up the heat by introducing more sources of failure. In addition to encoding this algorithm, we also verify its correctness using classical process-theoretic coinductive reasoning techniques. This, too, is successful, except that we don't verify the full Two Phase Commit Protocol, only a special case. This is less of an omission then may appear at first sight, because the special case already contains all the difficult issues other

than the combinatorial explosion of trivial intermediate states. It is conceptually straightforward to rewrite our proofs to verify the full protocol, but the price in terms of notational complexity was deemed too high. This inability to compress all those trivial intermediate states into something manageable shows that something fundamental is still missing in the author's understanding of the mathematics of distributed computation.

1.5 Structure of this Text

The next chapter introduces the π-calculus and its key reasoning tools. In Chapter 3 we explain the basic ideas behind the Two Phase Commit Protocol and sketch its proof assuming the absence of failure. Chapters 6 and 8 later will elaborate on this proof by introducing failures. It may be interesting for the reader to see how additional failures complicate parts of the proof. Chapter 4 introduces timers and studies the properties of the resulting calculus in some detail. In Chapter 7, we add sites and message failure to the timed π-calculus of Chapter 4 and study the effect this has on the reduction congruence. In Chapter 7 sites find a second use when we introduce site failure and persistence. Chapter 9 closes this thesis.

Chapter 2

Basic π-Calculi

This chapter gives an overview of π-calculi with particular emphasis on concepts that will be important later on. Basic knowledge of λ-calculi [11] and models of concurrency is helpful [17]. More comprehensive accounts of π-calculi can be found in [77, 103].

2.1 Introduction

In the first half of the 20th century, various formalisms were proposed to capture the informal concept of computable function, μ-recursive functions, Turing Machines and the λ-calculus possibly being the most well-known examples today [11, 31, 51, 87]. The surprising fact that they are essentially equivalent in the sense that they are all encodable into each other is the content of the *Church-Turing Thesis* [28, 51]. Another shared feature is more rarely commented on: they all are most readily understood as models of *sequential* computation. With hindsight that is surprising: after all, sequential computation can be understood as a peculiar form of concurrency [16, 75]. In addition, paradigmatic forms of message passing that could have served as an inspiration, such as the telephone system, were already deployed in the western world at the time when the pioneering theorists of computability contributed their insights. Whatever may have caused this early emphasis on sequentiality, in the end, the mathematical and philosophical problems that sparked the search for formalisms could be solved entirely within the sequential world.

The subsequent consolidation of computer science required a more subtle formulation of the notion of computation, in particular explicit representations of concurrency and communication. Petri-Nets [94, 95] and process calculi such as CSP [52], CCS [74] and π-calculi are currently the most prominent calculi to have emerged from this line of enquiry.

The process calculus approach gathered momentum in the 1970s when it became increasingly clear that the then dominant functional approach to modelling

computation [48] was unlikely to yield satisfactory accounts of non-deterministic, non-terminating and interacting agents. Instead, early pioneers, such as Milner and Hoare decided to make interaction between agents executing in parallel the basic computational primitive. This can be done in several ways, but the CSP and CCS approach, from which π-calculi descend, can be broadly characterised by three core design decisions.

- Computing agents have zero or more discrete points of connection and interaction called, interchangeably, *names, channels* or *interaction points*. This can be graphically represented. Figure 2.1 gives an example of a process with 3 interaction points x, y and z. Parallel composition of two agents involves connecting their interaction points by *links*, whenever they share a name (cf. Figure 2.2) [56]. Crucially, a link does not preclude further connections at a name. The Internet is a good source of analogies here: names correspond to IP addresses (plus port numbers, but let's ignore this detail for simplicity). For this analogy to work, the interaction points of a process are all the IP addresses it can receive data on *and* all the IP addresses it uses to send data to. Because computing agents can be conceptualised as simple 'geometric' objects, as in Figures 2.1 and 2.2, this approach is sometimes referred to as the "pineapple approach to concurrency".

- Computation itself is binary, point-to-point interaction between independent agents. Interaction happens along names by handshaking or synchronisation between a sender and a receiver. This handshaking may or may not involve passing data from the sender to the receiver. In the Internet, interaction happens by sending IP packets between computers.

- Interaction is an atemporal event it the sense that it does not have a duration. Interactions may be ordered in time. Here the Internet analogy breaks down because the duration of packet delivery from sender to receiver has important semantic consequences.

With this in mind, we can sketch the process calculus approach to modelling concurrency, as exemplified by π-calculi.

To define a process calculus, one starts with a set of *names*, the discrete interaction points. Names have no internal structure apart from what is required to distinguish names from one another. Hence names are *pure* in the sense of Needham [83]. Their only purpose is to denote interaction points. In many implementations, names have rich internal structure to improve efficiency, but this is abstracted away in almost all process theoretic models. With points of interaction fixed we must have means to form new processes from old. In other words, we need an algebra of processes. The crucial algebraic operators, always present in some form or other, allow

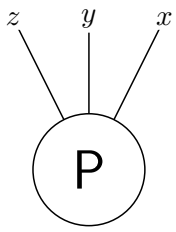


Figure 2.1: A process P with three interaction points x, y and z.

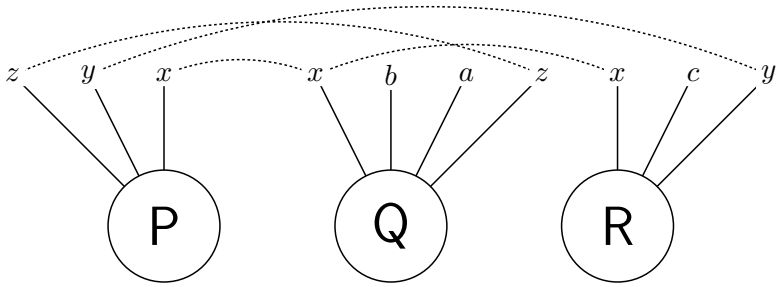


Figure 2.2: Three processes, P, Q and R composed in parallel. Their shared names x and y are still available for further connections. The dotted lines represent channels, paths for potential flow of information between these processes.

- the parallel composition of processes,
- to specify which channels to use for sending and receiving data,
- sequentialisation of interactions,
- hiding of interaction points and
- recursion or replication.

Parallel composition of two processes P and Q, usually written $\mathsf{P} \mid \mathsf{Q}$ is the key primitive distinguishing sequential models of computation from process calculi. It allows computation in P and Q to proceed simultaneously and independently. But it also allows interaction, that is synchronisation and flow of information from P to Q on a channel shared by both (or vice versa). Channels are *created* by shared names in parallel composition. Clearly, $\mid$ is an associative and commutative operation.

Interaction in all the calculi we shall consider is a *directed* flow of information. That means, input and output are distinguished as dual interaction primitives. We have an input operator $x(v)$ or just x and an output operator $\overline{x}\langle y\rangle$ or $\overline{x}$, all of which name an interaction point (here x) that is used to synchronise with a dual interaction primitive. This name is called *subject*. Should information be exchanged, it will flow from the outputting to the inputting process. The output primitive will specify the data to be sent. In $\overline{x}\langle y\rangle$, this data is y and it is called the *object* in $\overline{x}\langle y\rangle$. Similarly, if an input expects to receive data, one or more *bound variables* will act as place-holders to be substituted by data, when it arrives. In $x(v)$, v plays that role. But what kind of data is exchanged in an interaction? There are various choices. It will turn out that this choice is a key distinguishing feature between process calculi.

Sometimes interactions must be temporally ordered, because we might want to specify algorithms like "first receive some data on x and then send that data on y". Sequential composition can be used for such purposes. It is well-known from other models of computation. In process calculi, the sequentialisation operator is usually integrated with input or output or both. For example the process $x(v).\mathsf{P}$ will wait for an input on x. Only when this input occurs, the *continuation* P will be activated with the received data substituted for v.

The key operational rule – comparable to β-reduction in λ-calculi, containing the computational essence of process calculi, can be given solely in terms of parallel composition, sequentialisation, input and output: although the details vary, it always looks something like this.

$$(\textsc{Com}) \quad \overline{x}\langle y\rangle.\mathsf{P} \mid x(v).\mathsf{Q} \;\rightarrow\; \mathsf{P} \mid \mathsf{Q}\{y/v\}$$

The process $\overline{x}\langle y\rangle.\mathsf{P}$ sends a message, here y, along the channel x. Once that message has been sent, $\overline{x}\langle y\rangle.\mathsf{P}$ becomes the process P. P is the *continuation* of $\overline{x}\langle y\rangle.\mathsf{P}$. Dually,

the process $x(v).\mathsf{Q}$ receives that message on channel x to become $\mathsf{Q}\{y/v\}$, which is Q with the place-holder v substituted by y, the data received on x. The class of processes that P is allowed to range over as the continuation of the output operation substantially influences the properties of the calculus.

It may be instructive to compare (Com) with

$$(\beta) \quad (\lambda x.M)N \rightarrow M\{N/x\},$$

the corresponding rule of λ-calculi [11]. In both cases, some data is distributed in the target term. In processes, y is replacing v in Q whereas for λ-calculi, N replaces x in M. The crucial difference is that y is *always* a trivial entity, devoid of internal structure apart form what is guaranteeing its identity, while N can be arbitrarily complex. It can be shown that β-reduction can be decomposed into several (Com) operations in a clean and natural fashion [16, 75, 103]. Converse encodings have not been considered much. Those that have been proposed are cumbersome and lack various desirable mathematical properties such as compositionality (cf. §2.4). This asymmetry is likely to be one of the key reasons for the better expressivity of (some) process calculi, in comparison with λ-calculi.

Regarding hiding, as already pointed out, the view of agents as geometric objects with discrete interaction points does not limit the number of connection that can be made at a given interaction point. Figure 2.2 gave an example of multiple channels at one port. But interaction points allow interference (i.e. interaction). For the synthesis of compact, minimal and compositional systems, the ability to restrict interference is crucial. *Hiding* operations allow to control the connections made between interaction points when composing agents in parallel. In sequential models of computation, scoping rules, procedures and objects facilitate hiding (but they might have other uses, too: functional features in the case of procedures and subtyping with its associated dispatch mechanisms for objects). We denote the hiding of a name x in P by $(\nu x)\mathsf{P}$. Figure 2.3 shows the effect of going from P to $(\nu x)\mathsf{P}$. The process P on the left can interact with the outside world on x, y and z. In contrast, $(\nu x)\mathsf{P}$ on the right can only use y and z for this purpose. The restriction does not prevent usage of x inside P. But what happens if x gets sent to a process outside of $(\nu x)\mathsf{P}$, as may happen in $(\nu x)(\overline{y}\langle x\rangle \mid \mathsf{Q})$, provided $x \neq y$? Whether or not it is possible to communicate a name hidden this way is another important point of divergence between different calculi.

The operations presented so far allow to describe only finite interaction and are consequently insufficient for full computability, which includes non-terminating behaviour. *Recursion* or *replication* are operations that allow finite descriptions of infinite behaviour. Recursion is well-known from the sequential world. Replication $!\mathsf{P}$ can be understood as abbreviating $\mathsf{P} \mid \mathsf{P} \mid \mathsf{P}...$, the parallel composition of a countably infinite number of P's. Replication is often more convenient than recursion,

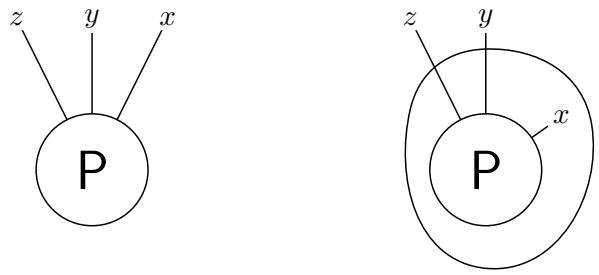


Figure 2.3: A geometric view of the effect of restricting x in P. The process on the right is $(\nu x)\mathsf{P}$.

but either can express the other.

So far all algebraic operators allowed to produce new processes from old, but how do we get off the ground? The most popular solution is a *null process* which has no interaction points. It is utterly inactive and its sole purpose is to act as the inductive anchor on top of which we can generate more interesting processes.

The set of *processes* is usually given as a term algebra. Once the syntax of a process calculus has been defined, its *semantics* or *dynamics* are given by a binary *reduction relation* $\rightarrow$ on processes, induced by the basic rules of interaction, as sketched above. In addition to the operators introduced so far, calculi may of course also have other algebraic operations, often with serious semantic consequences.

This concludes the abstract account of what we understand as processes. Of course many more details need to be specified for any particular process model. We finish this section we summarise the key factors that roughly distinguish between important classes of process calculi.

- What kind of data is exchanged in interactions?
- How do hiding of interaction points and communication relate?
- What kind of continuations is the output operation allowed to have?

2.2 π-Calculi

Having sketched the key characteristics of process calculi, we now present its most well-studied instance, π-calculi. The first thing to know about π-calculi is that they are name passing formalisms. That means the data exchanged is names and names only. Since names are in some ways the only kind of data that *needs* to be present in a process calculus, π-calculi give the simplest possible answer to our question "What kind of data is exchanged in interactions". As names are exchanged in interactions, our second question also becomes pertinent: can restricted names

be communicated to the outside? The answer is an emphatic "yes": π-calculi exhibit *scope mobility*. What that means is explained below, because it is easier to describe after the syntax of π-calculi has been clarified. It has been quite a surprise that despite the simplistic nature of the data under exchange, π-calculi are among the most expressive formalisms. Scope mobility is the key novelty distinguishing π-calculi from their predecessors. The third question above is orthogonal to the concept of a π-calculus, in the sense that different π-calculi give drastically different answers.

We begin with the asynchronous π-calculus because it is probably the simplest variant. Later we describe, albeit more briefly, other π-calculi, partly because contrasting variants might aid understanding and because some of the variants will be useful in our development of abstractions for DS, but also to communicate the current lack of canonicity in π-calculi.

2.2.1 The Asynchronous π-Calculus

The asynchronous π-calculus was first introduced independently by Honda and Tokoro [54, 58, 59] and by Boudol [22]. It is a simplification of the original π-calculus of Milner, Parrow and Walker [78] both, in its syntactic presentation and in its mathematical properties. The aim of its development was to extract the very essence of name passing: to have a calculus as expressive as the original π-calculus, but without redundant operations. Because it in some respects the most simple π-calculus possible, it will be our reference calculus throughout this text.

But what is asynchrony? This term is often explained with reference to interacting systems where messages are exchanged via a medium (such as an explicit networking layer) that decouples senders from receivers: for example, if the medium acts like an unbounded buffer, and that is quite a realistic abstraction of exchanging messages over the Internet, the delay between sending a message and its receipt may grow arbitrarily large. Selinger [106] axiomatises this notion of asynchrony for CCS-like calculi at the level of semantics. If the delay between sending a message and its receipt cannot be bounded, it is probably not a good idea to block senders until receipt of their messages. This, then, is asynchrony: that senders are not *directly* affected by the receipt (or otherwise) of their messages. Of course receivers may send explicit acknowledgements back to senders which will change the original senders, but in this case it is the receipt of the acknowledgement message, not the receipt of the original message that institutes the change.

To make this more formal, assume a countably infinite set of names, ranged over by $v, w, z, y, z, a, b...$ The syntax of the asynchronous π-calculus is straightforward

and given by the following grammar.

P	::=	$\overline{x}\langle\vec{y}\rangle$	output
	|	$x(\vec{y}).\mathsf{P}$	input prefix
	|	$!x(\vec{y}).\mathsf{P}$	lazy replication
	|	$\mathsf{P} \mid \mathsf{Q}$	parallel composition
	|	$(\nu x)\mathsf{P}$	hiding
	|	$\mathsf{0}$	inaction

Apart from the parallel composition $\mathsf{P} \mid \mathsf{Q}$ of processes P and Q, which works exactly as described in the introduction, we have 5 other constructors which we shall now present in some (more) detail.

Input prefixing $x(\vec{y}).\mathsf{P}$ is really an abbreviation for $x(y_0...y_{n-1}).\mathsf{P}$ with $n \geq 0$ and adds a discrete interaction point x to those of P which may or may not already contain x. The process $x(\vec{y}).\mathsf{P}$ waits on x for an interaction with a corresponding output, blocking its *continuation* P until such an interaction happens. When it happens, the process receives a tuple $\vec{z}$ of names which will be substituted in P for $\vec{y}$ (the substitution operation will be defined below). The names $y_0, ..., y_{n-1}$ are *bound* in $x(\vec{y}).\mathsf{P}$, hence input prefixing will in general remove free names. Input prefixing is the *only* form of sequencing the asynchronous π-calculus offers. We will usually write $x.\mathsf{P}$ for $x().\mathsf{P}$. We allow tuples $\vec{y}$ to be used for convenience and generality. We can restrict to the *monadic* asynchronous π-calculus, where all tuples are of length 1. This does not significantly affect the available computations, cf. §2.4.3.

The process $\overline{x}\langle\vec{y}\rangle$, short for $\overline{x}\langle y_0...y_{n-1}\rangle$ with $n \geq 0$, is called *free output* or just *output*. The process sends a message, containing $\vec{y}$ as objects, along the subject channel x. All names $x, y_0...y_{n-1}$ are *free*. That means they are the interaction points of the process $\overline{x}\langle\vec{y}\rangle$. We will discuss below, why it makes sense to consider the $\vec{y}$ interaction points, even though $\overline{x}\langle\vec{y}\rangle$ cannot itself send or receive messages on any y_i in $\vec{y}$. Output does not involve sequencing as it does not have a continuation: we cannot form the process $\overline{x}\langle\vec{y}\rangle.P$. Alternatively, we may say that $\mathsf{P} = \mathsf{0}$ is the only legal continuation for $\overline{x}\langle\vec{y}\rangle.\mathsf{P}$. This is how asynchrony is achieved in the asynchronous π-calculus. It will become clear later when we present the semantics of this calculus, that the lack of an output continuation means, a process can never be affected by the receipt (or otherwise) of messages it emitted. We abbreviate $\overline{x}\langle\rangle$ to $\overline{x}$.

Lazy replication $!x(\vec{v}).\mathsf{P}$ is an instance of the more general replication, sketched in the previous section and $!x(\vec{v}).\mathsf{P}$ can be considered the parallel composition of infinitely many copies of $x(\vec{v}).\mathsf{P}$. Unrestricted replication $!\mathsf{P}$ is another possible syntactic choice here, but it scales less gracefully to DS, as we will detail in Section 2.4.2 and Chapter 4.

$\mathsf{0}$ is the *inactive* or *inert* process. It cannot perform any computation. It is often omitted when immediately preceded by an input: so we will usually write $x(\vec{v})$ rather than $x(\vec{v}).\mathsf{0}$.

The remaining operation, *hiding* of a name x in P is achieved by $(\nu x)\mathsf{P}$, will be explained after the semantics of the asynchronous π-calculus has been defined.

Free Names, Bound Names

We have repeatedly emphasised the importance of names or interaction points for processes. We defined them to be the places that *may* be used for interference (ie. interaction), that is, for sending or receiving messages. Now that we have the syntax of the asynchronous π-calculus nailed down, we can be precise about the names of a process: the sets $\mathsf{fn}(\mathsf{P})$ of *free names* of P and $\mathsf{bn}(\mathsf{P})$ of *bound names* of P are given inductively by the following clauses.

$$
\begin{array}{rcl@{\qquad}rcl}
\mathsf{fn}(\overline{x}\langle\vec{y}\rangle) & = & \{x,\vec{y}\} & \mathsf{bn}(\overline{x}\langle\vec{y}\rangle) & = & \emptyset \\
\mathsf{fn}(x(\vec{v}).\mathsf{P}) & = & \{x\}\cup(\mathsf{fn}(\mathsf{P})\setminus\{\vec{v}\}) & \mathsf{bn}(x(\vec{v}).\mathsf{P}) & = & \mathsf{bn}(\mathsf{P})\cup\{\vec{v}\} \\
\mathsf{fn}(!x(\vec{v}).\mathsf{P}) & = & \{x\}\cup(\mathsf{fn}(\mathsf{P})\setminus\{\vec{v}\}) & \mathsf{bn}(!x(\vec{v}).\mathsf{P}) & = & \mathsf{bn}(\mathsf{P})\cup\{\vec{v}\} \\
\mathsf{fn}(\mathsf{P}\mid\mathsf{Q}) & = & \mathsf{fn}(\mathsf{P})\cup\mathsf{fn}(\mathsf{Q}) & \mathsf{bn}(\mathsf{P}\mid\mathsf{Q}) & = & \mathsf{bn}(\mathsf{P})\cup\mathsf{bn}(\mathsf{Q}) \\
\mathsf{fn}((\nu x)\mathsf{P}) & = & \mathsf{fn}(\mathsf{P})\setminus\{x\} & \mathsf{bn}((\nu x)\mathsf{P}) & = & \mathsf{bn}(\mathsf{P})\cup\{x\} \\
\mathsf{fn}(\mathsf{0}) & = & \emptyset & \mathsf{bn}(\mathsf{0}) & = & \emptyset
\end{array}
$$

We sometimes write $\mathsf{P}\bowtie\mathsf{Q}$ to indicate that $\mathsf{fn}(\mathsf{P})\cap\mathsf{fn}(\mathsf{Q})=\emptyset$. "*Free names*" is yet another term for names or interaction points.

What is the rationale behind this definition? For example, why is $y\in\mathsf{fn}(x(v).\overline{y}\langle v\rangle)$? The process $x(v).\overline{y}\langle v\rangle$ can neither send nor receive data on y (if $x\neq y$). All it can do is input on x. Of course by receipt of, say, a at x, $x(v).\overline{y}\langle v\rangle$ evolves into $\overline{y}\langle a\rangle$, but $\overline{y}\langle a\rangle$ is not $x(v).\overline{y}\langle v\rangle$! And what about y being free in $\overline{x}\langle y\rangle$? There are two reasons for our definition of $\mathsf{fn}(\cdot)$.

1. We want interaction points to be "stable under computation": $\mathsf{P}\twoheadrightarrow\mathsf{Q}$ must imply $\mathsf{fn}(\mathsf{Q})\subseteq\mathsf{fn}(\mathsf{P})$. This means we see free names as a form of specification of process behaviour: no matter what P does, it will never receive or send data via names that are not in $\mathsf{fn}(\mathsf{P})$.

2. To keep the mathematics simple, we prefer $\mathsf{fn}(\mathsf{P})$ to be defined by induction on the structure of P.

Without (2), it would be possible, for example, to have $\mathsf{fn}(\overline{x}\langle y\rangle)=\{x\}$, but $\mathsf{fn}((\nu x)(\overline{x}\langle y\rangle\mid x(v).\overline{v}))=\{y\}$, at the expense of a significantly more complicated definition of $\mathsf{fn}(\cdot)$. It is conceivable that different accounts of what counts as free names are possible, but so far they have not been forthcoming.

2.3 Semantics

The semantics of process calculi – their computational essence – can be presented in various ways. We focus on *reductions* and *labelled transitions*. They are most elementary in the sense that they require the least amount of mathematical machinery, and are currently the most popular tools for this purpose.

Reductions, often denoted $\rightarrow$, are binary relations on processes. A *reduction* $\mathsf{P} \rightarrow \mathsf{Q}$ should be understood as specifying an atomic computational step. We say that P *evolves into* Q or that Q is a *one step reduct* of P if $\mathsf{P} \rightarrow \mathsf{Q}$. In process calculi, such atomic steps would usually involve the exchange of a message between two processes, but alternatives are possible: for calculi with functional features, β-reduction could count as atomic [11] and imperative features could be modelled with state change and conditional branches [92, 119]. In calculi of mobile computation, agent migration between locations would feature in this role [25].

While reduction based accounts of computation are robust, meaning that it is almost always straightforward to use reductions to formalise whatever notion of computation one has in mind, they sometimes make it a bit cumbersome to define the *observables* of processes. Observables are crucial as soon as one wants to answer questions like "are P and Q the same" or "can I replace P by Q without changing the semantics"? The observables of a process P describe what kind of interactions P offers to the environment. This is where labelled transitions shine. While usually more difficult to set up, once in place, they define the computations of a process by the observations an observer can make of the process. A transition $\mathsf{P} \stackrel{l}{\rightarrow} \mathsf{Q}$ can be read as "the process P allows an observer to make the observation l and if the observation takes place, P evolves into Q". This view of observations assumes that observers are themselves processes and posits an intimate connection between observations and atomic computational steps. In practise one needs both: reductions to have a firm and straightforward notion of semantics and labelled transitions for a theory of observation and process equivalence to allow efficient reasoning. Unfortunately, a mechanical way of going from reductions to labelled transitions has not been forthcoming, except for restricted classes of processes [67, 107].

2.3.1 Dynamics

We specify the reduction semantics of the asynchronous π-calculus using *structural congruences*. This approach was pioneered by Berry's and Boudol's Chemical Abstract Machine [20] and first applied to π-calculi by Milner [75].

Structural Congruence

To allow convenient presentation, structural congruences reduce the rigidity of the syntax of processes by equating processes that are computationally equivalent but

have distinct syntactic representations. The reduction relation must be closed under the chosen structural congruence. An example of a distinction introduced by the syntax would be that between $\mathsf{P} \mid \mathsf{Q}$ and $\mathsf{Q} \mid \mathsf{P}$, which should behave exactly the same in every conceivable semantics of processes.

The *structural congruence* $\equiv$ is the smallest binary congruence over terms satisfying the following axioms.

1. If $\mathsf{P} \equiv_\alpha \mathsf{Q}$ then $\mathsf{P} \equiv \mathsf{Q}$.

2. $\mathsf{P} \mid \mathsf{Q} \equiv \mathsf{Q} \mid \mathsf{P}$,

3. $\mathsf{P} \mid (\mathsf{Q} \mid \mathsf{R}) \equiv (\mathsf{P} \mid \mathsf{Q}) \mid \mathsf{R}$,

4. $(\nu x)(\nu y)\mathsf{P} \equiv (\nu y)(\nu x)\mathsf{P}$,

5. $(\nu x)\mathsf{0} \equiv \mathsf{0}$,

6. $\mathsf{P} \mid \mathsf{0} \equiv \mathsf{P}$.

7. $\mathsf{P} \mid (\nu x)\mathsf{Q} \equiv (\nu x)(\mathsf{P} \mid \mathsf{Q})$ if $x \notin \mathsf{fn}(\mathsf{P})$,

Axiom 1 formalises that we do not distinguish between processes that are α-convertible. The next two axioms, 2 and 3, make parallel composition into an associative and commutative operation. Restriction becomes a "commutative" operation by Axiom 4. Axioms 5 and 6 are the *garbage collection axioms*. They allow to remove some simple processes that cannot do any computation. Some processes that cannot participate in computations cannot be removed with $\equiv$, however: deadlocked processes that still contain input or output primitives, for example, are deemed too complicated to warrant garbage collection by the structural congruence. Finally, Axiom 7 allows *scope extension*, an operation that distinguishes π-calculi from most of their predecessors. Scope extension allows to enlarge the set of processes permitted to use a given hidden name. In particular, it allows to *dynamically create private channels* between processes. The additional expressivity that distinguish π-calculi from CCS and related models of concurrency resides mainly in this ability to dynamically create private links. Scope extension will be described in more detail shortly.

Not all the axioms are equally important for the computational content of the asynchronous π-calculus, for example the garbage collection axioms can be dropped. Other variants are possible.

Reductions

The reduction relation we shall be using is defined inductively on processes by the following five rules.

$$(\textsc{Com}) \quad \frac{\mathsf{length}(\vec{v}) = \mathsf{length}(\vec{y})}{x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \;\rightarrow\; \mathsf{P}\{\vec{y}/\vec{v}\}}$$

$$(\textsc{Rep}) \quad \frac{\mathsf{length}(\vec{v}) = \mathsf{length}(\vec{y})}{!x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \rightarrow !x(\vec{v}).\mathsf{P} \mid \mathsf{P}\{\vec{y}/\vec{v}\}}$$

$$(\textsc{Par}) \quad \frac{\mathsf{P} \rightarrow \mathsf{P}'}{\mathsf{P} \mid \mathsf{Q} \rightarrow \mathsf{P}' \mid \mathsf{Q}}$$

$$(\textsc{Res}) \quad \frac{\mathsf{P} \rightarrow \mathsf{Q}}{(\nu x)\mathsf{P} \rightarrow (\nu x)\mathsf{Q}}$$

$$(\textsc{Cong}) \quad \frac{\mathsf{P} \equiv \mathsf{P}' \quad \mathsf{P}' \rightarrow \mathsf{Q}' \quad \mathsf{Q}' \equiv \mathsf{Q}}{\mathsf{P} \rightarrow \mathsf{Q}}$$

The (Com) and (Rep) rules contain the computational essence of the asynchronous π-calculus: a process waiting for an input on channel x can receive a tuple of names from another process, running in parallel, that sends names along x, provided the length of the tuple of names to be sent coincides with the length of the tuple of names to be received. The only difference between the two is that in the (Rep) rule, the receiver stays in the configuration unchanged. It spawns off a process to receive the data from the sender while in (Com), the receiver evolves into its continuation with a suitable substitution of the received values for their place-holders. The (Par) rule and the (Res) rule allow to infer computations taking place under restrictions or in processes running in parallel with other processes. Naturally, neither restriction nor parallel composition limit computation taking place *inside* subprocesses. Finally, the (Cong) rule connects the structural congruence and $\rightarrow$ by closing the latter under $\equiv$. This keeps the axioms for $\rightarrow$ simple.

Coming back to the question of asynchrony, if Q is an arbitrary process, then

$$(\overline{x}\langle y\rangle \mid \mathsf{Q}) \mid x(v).\mathsf{P} \rightarrow \mathsf{Q} \mid \mathsf{P}\{y/v\}$$

(and similarly for interaction with a repeated input). So the consumption of $\overline{x}\langle y\rangle$ has no effect on the rest of $(\overline{x}\langle y\rangle \mid \mathsf{Q})$. Hence the calculus is indeed asynchronous.

Interestingly, the rules do *not* allow to infer $\mathsf{P} \mid \mathsf{Q} \rightarrow \mathsf{P}' \mid \mathsf{Q}'$ from $\mathsf{P} \rightarrow \mathsf{P}'$ and $\mathsf{Q} \rightarrow \mathsf{Q}'$, in other words, our semantics does not allow to have more than one interaction at a time. This means that our model accounts only for concurrent computation but not for (real) parallelism (also termed "true concurrency"). There are two reasons for this restriction.

- Almost all phenomena, in particular all those we are trying to model in this text, that make parallel computation interesting and difficult, already show up in concurrent computation.
- Models of parallelism are much more involved than models of concurrency and much less mathematically tractable.

Hence we do not loose anything with the restriction to concurrency but gain feasibility. Nevertheless, models of parallelism are a fascinating topic and we refer the interested reader to [36, 44, 45, 86, 118, 120] for further information.

Hiding and Scope Extension

We are now ready to explain hiding and scope extension in the asynchronous π-calculus. It works in the same way in all other π-calculi. We have already sketched that the operator (νx) restricts the outside from sending messages to P in $(\nu x)\mathsf{P}$. This is enforced by the definition of $\rightarrow$. In (Com) and (Rep), the two communicating processes do not feature a restriction guarding the inputting or outputting party. With that as base, we can show by induction on the derivation of reduction steps, that communication on a channel x never crosses a restriction (νx). The interesting issue is what happens when a restricted name wants to be communicated outside the scope of the restriction, as exemplified by the following processes, assuming $x \neq y$.

$$(\nu y)(\overline{x}\langle y\rangle \mid y(v).\mathsf{P}) \mid x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q})$$

Here $(\nu y)(\overline{x}\langle y\rangle \mid y(v).\mathsf{P})$ has the channel y as a private resource, but would like to communicate it to $x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q})$, maybe to allow P and Q to interact without interference from the outside. But (Com) cannot be applied because (νy) is in the way. Can we find a way around this? Yes, we use *scope extension*:

$$(\nu y)(\overline{x}\langle y\rangle \mid y(v).\mathsf{P}) \mid x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q}) \equiv (\nu y)(\overline{x}\langle y\rangle \mid y(v).\mathsf{P} \mid x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q}))$$

assuming that $y \notin \mathsf{fn}(x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q}))$. The scope of (νy) has been highlighted. Now we can infer

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\overline{x}\langle y\rangle \mid x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q}) \rightarrow \overline{y}\langle a\rangle \mid \mathsf{Q}\{y/v\}}}{\overline{x}\langle y\rangle \mid x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q}) \mid y(v).\mathsf{P} \rightarrow \overline{y}\langle a\rangle \mid \mathsf{Q}\{y/v\} \mid y(v).\mathsf{P}}}{(\nu y)(\overline{x}\langle y\rangle \mid x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q}) \mid y(v).\mathsf{P}) \rightarrow (\nu y)(\overline{y}\langle a\rangle \mid \mathsf{Q}\{y/v\} \mid y(v).\mathsf{P})}}{(\nu y)(\overline{x}\langle y\rangle \mid y(v).\mathsf{P}) \mid x(v).(\overline{v}\langle a\rangle \mid \mathsf{Q}) \rightarrow (\nu y)(\overline{y}\langle a\rangle \mid \mathsf{Q}\{y/v\} \mid y(v).\mathsf{P})}$$

where the last step uses (Cong). Figure 2.4 give a graphic representation of scope extension. What this shows is that in π-calculi, *private* links can be *dynamically* generated between processes. Although we don't understand expressiveness of programming languages well enough to dare a formal statement, this ability is likely

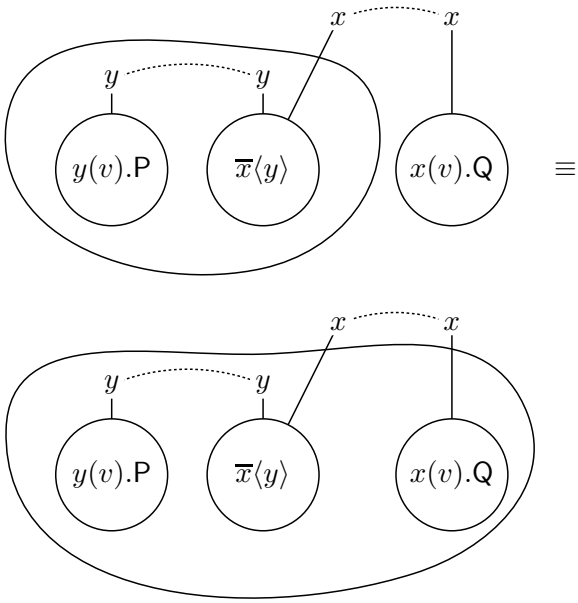


Figure 2.4: The effect of extending the scope of y.

to be the key ingredient making π-calculi much more expressive than their predecessors.

We often abbreviate $(\nu x_0)...(\nu x_{n-1})\mathsf{P}$ to $(\nu\vec{x})\mathsf{P}$ or $(\nu x_0...x_{n-1})\mathsf{P}$ and, if $S = \{x_0, ..., x_{n-1}\}$, $(\nu S)\mathsf{P}$ as a shorthand for $(\nu x_0...x_{n-1})\mathsf{P}$, understanding that $(\nu\emptyset)\mathsf{P} \equiv (\nu\langle\rangle)\mathsf{P} \equiv \mathsf{P}$.

Variable Convention and α-Convertibility

Having already used α-convertibility, we now briefly explain what it is. Two processes are *α-convertible* if one can be obtained from the other by renaming of bound names. For example

$$(\nu x)(\overline{x}\langle y\rangle \,|\, x(v).\overline{v}) \qquad \text{and} \qquad (\nu a)(\overline{a}\langle y\rangle \,|\, a(w).\overline{w})$$

are α convertible. This identification corresponds to equating $f(x) = x^2$ with $f(y) = y^2$ or $\sum_{i=1}^{m} a_i$ with $\sum_{j=1}^{m} a_j$ in classical mathematics. We shall not formally define the α-equality relation and refer the reader to [54] for the straightforward but tedious details. By $\equiv_\alpha$ we denote the smallest congruence equating all α-convertible terms.

Throughout the rest of this thesis we shall make liberal – and often tacit – use of the *variable convention*: in each process we consider, all occurring free names are assumed to be distinct from all occurring bound names [11]. It is easy but tedious and not very enlightening to adapt all proofs and definitions so that they would work even in the absence of the variable convention.

Substitution

One of the key ingredients of computation in π-calculi is substitution of names for names. Here is its definition. If $\vec{x} = \langle x_0, \ldots, x_{n-1}\rangle$ and $\vec{y} = \langle y_0, \ldots, y_{n-1}\rangle$ are two possibly empty tuples of names such that $i \neq j$ implies $y_i \neq y_j$ then the substitution $\{\vec{x}/\vec{y}\}$ is a unary function on processes defined as follows.

$$\begin{aligned}
\overline{x}\langle z_0, \ldots, z_{m-1}\rangle\{\vec{x}/\vec{y}\} &= \overline{x}\{\vec{x}/\vec{y}\}\langle z_0\{\vec{x}/\vec{y}\}, \ldots, z_{m-1}\{\vec{x}/\vec{y}\}\rangle \\
(x(\vec{v}).\mathsf{P})\{\vec{x}/\vec{y}\} &= x\{\vec{x}/\vec{y}\}(\vec{v}).(\mathsf{P}\{\vec{x}/\vec{y}\}) \\
(!x(\vec{v}).\mathsf{P})\{\vec{x}/\vec{y}\} &= !x\{\vec{x}/\vec{y}\}(\vec{v}).(\mathsf{P}\{\vec{x}/\vec{y}\}) \\
(\mathsf{P} \mid \mathsf{Q})\{\vec{x}/\vec{y}\} &= (\mathsf{P}\{\vec{x}/\vec{y}\}) \mid (\mathsf{Q}\{\vec{x}/\vec{y}\}) \\
((\nu x)\mathsf{P})\{\vec{x}/\vec{y}\} &= (\nu x)(\mathsf{P}\{\vec{x}/\vec{y}\}) \\
\mathsf{0}\{\vec{x}/\vec{y}\} &= \mathsf{0}
\end{aligned}$$

Substitution on names is straightforward.

$$x\{\vec{x}/\vec{y}\} = \begin{cases} x_i & \text{if } x = y_i \\ x & \text{if } x \neq y_i \text{ for all appropriate } i \end{cases}$$

Please note that this definition of substitution already uses the variable convention to ensure that in the clauses that deal with name binding operations (input prefix, lazy replication and restriction) all the bound names are distinct from all names occurring in $\{\vec{x}/\vec{y}\}$.

Contexts

For the development of the theory of π-calculi *n-holed linear (affine) contexts* are a convenient tool. Mathematically, they are term algebras over a set of variables, the n holes, such that each hole occurs exactly (at most) once. In process theory, it is customary to present contexts by way of a BNF-grammar and we do this for linear, one holed contexts.

$$C[\cdot] ::= [\cdot] \mid C[\cdot] \mid \mathsf{P} \mid \mathsf{P} \mid C[\cdot] \mid x(\vec{y}).C[\cdot] \mid (\nu x)C[\cdot] \mid !x(\vec{v}).C[\cdot]$$

Plugging a process P into a context $C[\cdot]$ is denoted $C[\mathsf{P}]$ and is a straightforward substitution operation, except that some of P's free variables might get *captured*: for example, if $C[\cdot] = \mathsf{P} \mid [\cdot] \mid \mathsf{R}$, then $C[\mathsf{Q}] = \mathsf{P}|\mathsf{Q} \mid \mathsf{R}$ but if $C[\cdot] = x(yz).[\cdot]$, then $C[\overline{x}\langle yyabz\rangle] = x(yz).\overline{x}\langle yyabz\rangle$. The generalisation to n-holed contexts is also straightforward: if $C[\cdot]_1[\cdot]_2 = (\nu x)([\cdot]_2 \mid [\cdot]_1)$, then $C[\mathsf{P}]_1[\overline{x}\langle yz\rangle]_2 = (\nu x)(\overline{x}\langle yz\rangle \mid \mathsf{P})$. We will often be concerned with subclasses of context, such as reduction contexts where holes are not allowed to be under a sequencing operator. An important subclass of n-holed linear (affine) contexts are *n-holed linear (affine) reduction contexts*

which are reduction contexts containing exactly (at most) one hole. For example, one-holed linear reduction contexts are generated by the following grammar.

$$C[\cdot] ::= [\cdot] \;\mid\; C[\cdot] \,|\, \mathsf{P} \;\mid\; \mathsf{P} \,|\, C[\cdot] \;\mid\; (\nu x)C[\cdot]$$

From now on, whenever we speak of *contexts* without further qualifications, they should be understood as one-holed linear contexts. Similarly, *reduction contexts* are one-holed linear reduction contexts.

Sorts and Sortings

The rules given so far do not prevent the formation of processes like

$$\mathsf{P} = \overline{x}\langle yz\rangle \,|\, x(abc) \,|\, \mathsf{Q}$$

which can be considered faulty because the process $x(abc)$ expects three names where a sender $\overline{x}\langle yz\rangle$ supplies only two. This phenomenon is called *arity mismatch*. In our account of the semantics of the asynchronous π-calculus, P is simply stuck, but it would also be possible to produce an explicit error instead [40].

In analogy with sequential languages [90], arity mismatch can be considered a consequence of *ill-typing* and *type systems* can be used to prevent it. The π-calculus community refers to processes that can exhibit arity mismatch as *ill-sorted* and typing systems that prevent ill-sorting as *sortings* or *sorts*. We concur with this naming tradition. Sortings are one of the simplest instances of typing systems for interacting processes and the only ones considered in this text.

Definition 1 [77] Given a set Σ of *sorts*, a *sorting* is a consists of an assignment of sorts to names and a partial function

$$\mathsf{sort} : \Sigma \rightharpoonup \Sigma^*.$$

We write $x : \sigma$ if the sort $\sigma \in \Sigma$ is assigned to the name x. If $y_i : \sigma_i$ for each y_i in $\vec{y}$ we write $\vec{y} : \vec{\sigma}$. A process *respects* a sorting sort if for each of its subterms $x(\vec{y}).\mathsf{Q}$ or $\overline{x}\langle\vec{y}\rangle$, whenever $x : \sigma$ then $\vec{y} : \mathsf{sort}(x)$. A process is *sortable* if it respects at least one sorting.

As an example of sorting, consider the sorts A, B and C and the sorting

$$A \mapsto (B, C) \qquad B \mapsto (A) \qquad C \mapsto ().$$

If we assign $x : A, y : B$ and $z : C$, then it sorts the process

$$x(yz).(\overline{y}\langle x\rangle \,|\, \overline{z}\langle\rangle) \qquad \text{but not} \qquad y(zx).(\overline{z}\langle y\rangle \,|\, \overline{x}\langle\rangle).$$

The relevant facts about sortings are summarised in the theorem below. Its proofs are scattered around the literature, cf. [116]

THEOREM 1 *1. Let* sort *be a sorting. If* $\mathsf{P} \rightarrow \mathsf{Q}$ *or* $\mathsf{P} \equiv \mathsf{Q}$ *and* P *respects* sort *then* Q *respects* sort*.*

2. Not every process is sortable.

3. Some processes are sortable by more than one sorting.

4. The set of all sortable processes is not recursive.

5. Sortable processes do not exhibit arity-mismatch.

Although we cannot mechanically decide whether a process is sortable, this is not a problem in practise as processes written by humans tend to adhere to straightforward sorting disciplines. In addition, sorting inference algorithms [40, 115] work well in many situations. Only with the introduction of second order types and generic interactions [16] sortings become somewhat challenging.

For the rest of this text, except where explicitly noted otherwise, we always assume all processes under consideration to be sortable. This allows to omit arity restrictions in communication rules. We will never present sortings explicitly because they are straightforward in all cases to be considered.

In addition, all equivalences and congruences on processes are assumed to be well-sorted. That means, they relate only well-sorted processes. This entails a weakening of the notion of congruence: for example, for $\mathcal{R}$ to be a binary, well-sorted, congruence, we require implications like $(\mathsf{P}_1, \mathsf{P}_2) \in \mathcal{R} \Rightarrow (\mathsf{P}_1 \mid \mathsf{Q}, \mathsf{P}_2 \mid \mathsf{Q}) \in \mathcal{R}$ to hold only if $\mathsf{P}_1 \mid \mathsf{Q}$ and $\mathsf{P}_2 \mid \mathsf{Q}$ are well-sorted.

Well-sortedness is a sufficient, but not a necessary requirement for the absence of arity-mismatches as the ill-sorted process

$$(\nu x)(\overline{x}\langle\vec{a}\rangle \mid x(v).(\overline{x}\langle bc\rangle \mid x(wu).\overline{v}\langle wu\rangle))$$

shows that reuses x, but in ways that prevent arity-mismatches.

2.3.2 Equivalences

We defined $\equiv$ to weaken the rigidity of our syntax, because we did not want to distinguish certain processes, despite syntactic differences, $\mathsf{P} \mid \mathsf{Q}$ and $\mathsf{Q} \mid \mathsf{P}$ being an example. But does $\equiv$ equate enough processes? One can show that $\mathsf{0} \not\equiv (\nu x)x.\overline{a}$, yet both processes are utterly inactive and cannot possibly influence any other computation. By this we mean, it is inconceivable that there could be a context $C[\cdot]$ where $C[\mathsf{0}]$ behaves differently from $C[(\nu x)x.\overline{a}]$. So $\equiv$ is still too fine a relation and we must look for something more appropriate.

This raises two related issues.

1. Given a binary relation $\approx$ on processes, how do we know it is right? In other words, what are the criteria that allow to judge that $\approx$ equates exactly those processes that cannot be distinguished computationally?

2. How do we define relations that could equate exactly the right processes, or, at least be a good approximation?

Unfortunately, neither question has been resolved conclusively so far. All we can do here is survey the state-of-the-art. Question (1) is philosophical rather than mathematical, because it relates formalisms with intuition. Hence certainty is unlikely to be forthcoming. Nevertheless, process theorists seem to have converged towards something like the following criterion: a candidate relation $\approx$ must at least satisfy $\mathsf{P} \approx \mathsf{Q}$ if and only if for all contexts $C[\cdot]$: $C[\mathsf{P}] \approx C[\mathsf{Q}]$. This formalises the requirement that two processes are equal exactly when no observers (considered as processes in the same calculus) can detect any difference. Detection of differences can only happen by interaction. Because this requirement only allows intrinsic observation of processes, that is observers are other processes in the same calculus and observations are computations, this approach to equivalences can be considered *atheistic*. It lacks omniscient outside observers who can make observations not available to the populace of the (computational) universe.

Anyway, this constraint is insufficient, because it is met by too many inappropriate relations, in particular by the identity and the universal relation on processes. Hence we need additional requirements. Here is a list of some proposals that a candidate relation $\approx$ could be asked to meet.

1. Consistency of $\approx$. That means $\mathsf{P} \not\approx \mathsf{Q}$ for at least some processes P and Q. Without consistency, the resulting equivalence would be trivial, but we know from bitter programming experience that concurrent computation is anything but.

2. Since process theorists are generally lazy – well, at least this one is – it would be nice if $\approx$ was maximum or at least maximal. Extremality simplifies reasoning.

3. If $\approx$ was a congruence, then that, too, would make reasoning about the calculus more modular and feasible.

4. We have some strong intuitions that some processes should be equated by $\approx$, $\mathsf{P} \mid \mathsf{Q}$ and $\mathsf{Q} \mid \mathsf{P}$ being our standard example. Hence $\equiv \subseteq \approx$ could be another requirement. In addition we'd probably also want to equate all processes P and Q that can never interact with other processes, such as $\mathsf{0}$ or $(\nu xy)(\overline{x} \mid \overline{y} \mid x.y.x.\overline{a})$. Such terms are called *insensitive* by [61] and will be important later.

5. If $\mathsf{P} \approx \mathsf{Q}$ and $\mathsf{P} \rightarrow\!\!\!\!\!\rightarrow \mathsf{P}'$ then there should be a process Q' with $\mathsf{Q} \rightarrow\!\!\!\!\!\rightarrow \mathsf{Q}'$ and $\mathsf{P}' \approx \mathsf{Q}'$. Otherwise $\approx$ would equate processes that could be said to compute differently. The technical term for this requirement is *reduction closure*.

6. A key parameter in process calculi is the notion of observable. Abstractly, observables are just unary predicates on processes, often called *barbs*. Not all unary predicates are computationally meaningful. We'd like a barb to be in correspondence with computational effects. That is, a process should allow a given observation exactly when some other process can make its behaviour contingent upon whether it can make that observation (by interaction) or not. For $\approx$ this means that $\approx$ must preserve observations: if $\mathsf{P} \approx \mathsf{Q}$ and P allows a certain observation to be made, then Q must also admit this observation.

Do we have to impose all these requirements at the same time? It is not clear how to answer this question in general, but the development below of the main equivalences for the asynchronous π-calculus indicates redundancies in the requirements (1) - (6).

Next we introduce the most popular equivalences for π-calculi. They can be roughly classified according to their answers to the following questions.

- How is congruency achieved?
- What kind of barbs do they preserve?
- What kind of equations are enforced by definition? This question relates to the definition of equivalences, which are often of the form "... $\mathcal{R}$ is the largest binary relation on processes such that $\mathcal{R}' \subseteq \mathcal{R}$ and ...". It is $\mathcal{R}'$ that is relevant here. For example, several equivalences are required to contain $\equiv$.
- How are the continuations of related processes related? This last point may require some explanation. Assume $\mathsf{P} \approx \mathsf{Q}$ and P (where $\approx$ is the equation under discussion) can do some computation to become P'. Or an observer can observe some of P's behaviour, in the course of which P evolves into P'. All equivalences we consider here, require Q to somehow match what P can do. This matching yields a process Q'. Do we always want $\mathsf{P}' \approx \mathsf{Q}'$ or does it sometimes make sense to be less restrictive?

We begin the journey through the exciting world of π-calculus equivalences with some very general definitions which will prove useful throughout this thesis, but in various restricted forms.

DEFINITION 2 A *congruence* is an equivalence relation $\mathcal{R}$ on processes such that $\equiv \subseteq \mathcal{R}$ and whenever $\mathsf{P} \ \mathcal{R} \ \mathsf{Q}$ then for all contexts $C[\cdot]$: $C[\mathsf{P}] \ \mathcal{R} \ C[\mathsf{Q}]$.

The inclusion of $\equiv$ into congruences is a mild deviation from the its conventional algebraic definition. It prevents congruences from being too sensitive to syntactic details.

DEFINITION 3 Given a set B of barbs, a symmetric binary relation $\mathcal{R}$ on processes is a *sound B-reduction theory* if $\mathcal{R}$ is a congruence and $(\mathsf{P},\mathsf{Q}) \in \mathcal{R}$ implies

- for every barb $b \in B$: if $b(\mathsf{P})$ then there is a reduction sequence $\mathsf{Q} \twoheadrightarrow \mathsf{Q}'$ such that $b(\mathsf{Q}')$,
- whenever $\mathsf{P} \twoheadrightarrow \mathsf{P}'$ then there is a reduction sequence $\mathsf{Q} \twoheadrightarrow \mathsf{Q}'$ such that $(\mathsf{P}',\mathsf{Q}') \in \mathcal{R}$.

The union of all sound B-reduction theories is easily seen to be a sound B-reduction theory itself and denoted $\overset{rc}{\approx}_B$. It is called *B-reduction congruence*.

One peculiarity of the previous definition is that computation sequences $\mathsf{P} \underbrace{\rightarrow \cdots \rightarrow}_{m} \mathsf{P}'$ are matched by $\mathsf{Q} \underbrace{\rightarrow \cdots \rightarrow}_{n} \mathsf{Q}'$ without constraints on m and n. One does not have to be so liberal.

DEFINITION 4 With B as in Definition 3, a symmetric binary relation $\mathcal{R}$ on processes is a *strong sound B-reduction theory* if $\mathcal{R}$ is a congruence and $(\mathsf{P},\mathsf{Q}) \in \mathcal{R}$ implies

- for every barb $b \in B$: if $b(\mathsf{P})$ then $b(\mathsf{Q})$,
- whenever $\mathsf{P} \rightarrow \mathsf{P}'$ then there is a reduction step $\mathsf{Q} \rightarrow \mathsf{Q}'$ such that $(\mathsf{P}',\mathsf{Q}') \in \mathcal{R}$.

The union of all strong sound B-reduction theories is easily seen to be a strong sound B-reduction theory itself and denoted $\overset{rc}{\sim}_B$. It is called *strong B-reduction congruence*.

How does (strong) B-congruence fare with respect to the above crude taxonomy of equivalences? Preservation of B-barbs is by definition and parametric on B. Continuations of related processes are only related for reduction sequences, *but not for barbs*. The only equations enforced by the definition are those given by $\equiv$. As to congruency, the definition of $\overset{rc}{\approx}_B$ and $\overset{rc}{\sim}_B$ is coinductive and achieves congruency by requiring all suitable pre-fixpoints (the (strong) sound B-reduction theories) to be congruences. One can argue that it is overkill to require congruency of *all* approximants to the largest equivalence, since we only use the maximum fixpoint for reasoning. It would be sufficient if that was a congruence. Milner's and Sangiorgi's classical definition of B-barbed congruence [79] is an example of such a more liberal construction.

DEFINITION 5 Given a set B of barbs, a symmetric binary relation $\mathcal{R}$ on processes is a *B-barbed bisimulation* if $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ implies

- for every barb $b \in B$: if $b(\mathsf{P})$ then there is a reduction sequence $\mathsf{Q} \rightarrow\!\!\!\!\rightarrow \mathsf{Q}'$ such that $b(\mathsf{Q}')$,
- whenever $\mathsf{P} \rightarrow\!\!\!\!\rightarrow \mathsf{P}'$ then there is a reduction sequence $\mathsf{Q} \rightarrow\!\!\!\!\rightarrow \mathsf{Q}'$ such that $(\mathsf{P}', \mathsf{Q}') \in \mathcal{R}$.

The union of all B-barbed bisimulations is a B-barbed bisimulations and denoted $\overset{be}{\approx}_B$. It is called *B-barbed equivalence*.

B-barbed congruence, denoted $\overset{bc}{\approx}_B$, is the largest congruence inside $\overset{be}{\approx}_B$:

$$\overset{bc}{\approx}_B = \{(\mathsf{P}, \mathsf{Q}) \mid \forall C[\cdot] : C[\mathsf{P}] \overset{be}{\approx}_B C[\mathsf{Q}]\}.$$

Strong B-barbed bisimulations, *strong B-barbed bisimilarity*, *strong B-barbed congruence*, $\overset{be}{\sim}_B$ and $\overset{bc}{\sim}_B$ are obtained analogously.

The only difference between (strong) B-barbed congruence and strong B-reduction congruence is the way congruency is achieved. For some important choices of barbs, $\overset{rc}{\approx}_B$ and $\overset{bc}{\approx}_B$ coincide, see Theorem 2.

Asynchronous Barbs

But what should barbs be? In π-calculi, an observer of a process P is another process Q, executing in parallel with P. It is not clear at all exactly what Q can or should observe: Q's observational power depends on just what kind of process it is allowed to be. Let's look at a simple scenario first: observers are arbitrary processes in the asynchronous π-calculus. Then an observer can receive messages from P and make its behaviour contingent upon just what it has received. Hence such observers can observe outputs of the process under observation; in particular, they can determine what channels it outputs on. On the other hand, as the receipt of a message cannot *directly* influence its sender in asynchronous calculi, asynchronous observers cannot observe inputs. The following definition is one way of characterising the essence of asynchronous observation as a unary predicate $\mathsf{P}\downarrow_x$.

DEFINITION 6 The *asynchronous barb* $\mathsf{P}\downarrow_x$ is defined inductively by the following rules.

$$\frac{}{\overline{x}\langle\vec{y}\rangle\downarrow_x} \qquad \frac{\mathsf{P}\downarrow_x}{\mathsf{P} \mid \mathsf{Q}\downarrow_x} \qquad \frac{\mathsf{P}\downarrow_x \quad x \neq y}{(\nu y)\mathsf{P}\downarrow_x}$$

Weak asynchronous barbs $\mathsf{P}\Downarrow_x$ are derived from asynchronous barbs as follows: $\mathsf{P}\Downarrow_x$ if and only if there is a reduction sequence $\mathsf{P} \rightarrow\!\!\!\!\rightarrow \mathsf{Q}$ such that $\mathsf{Q}\downarrow_x$. $\mathsf{P}\downarrow_x$ is to be read as: the process P can make an output on the channel x.

The rule involving the parallel combinator has a symmetric variant that has been omitted.

LEMMA 1 *If* $\mathsf{P} \equiv \mathsf{Q}$*, then* $\mathsf{P}\downarrow_x$ *implies* $\mathsf{Q}\downarrow_x$ *and* $\mathsf{P}\Downarrow_x$ *implies* $\mathsf{Q}\Downarrow_x$.

B-reduction congruence induced by taking B to be all asynchronous barbs coincides with that generated by weak asynchronous barbs. It is denoted $\overset{rc}{\approx}$ and called *reduction congruence*. Similarly, the B-barbed congruence induced by asynchronous barbs coincides with that induced by weak asynchronous barbs and is denoted $\overset{bc}{\approx}$. It is called *barbed congruence*. For quite a while it was an open question if reduction congruence and barbed congruence are the same relation, but in the asynchronous π-calculus, the matter is now settled in the most satisfactory manner.

THEOREM 2 *(Fournet and Gonthier [38])* $\overset{rc}{\approx} = \overset{bc}{\approx}$

Unfortunately, in some other π-calculi the two relations differ [104]. It is not clear what this means. The present author is inclined towards the opinion that this hints at the superiority of the asynchronous π-calculus. Sangiorgi and Walker argue [104] that reduction congruence is unnatural.

But why only take into account x in outputs $\overline{x}\langle\vec{y}\rangle$? We could also consider the names being transmitted:

$$\frac{}{\overline{x}\langle\vec{y}\rangle\downarrow_{\overline{x}\langle\vec{y}\rangle}}$$

(we omit the other cases as they are sufficiently similar to the definition of $\downarrow_x$, except that scope extensions must be taken into account, cf. [61]). The resulting congruences are less well understood and play no role for what we do in the subsequent chapters.

Synchronous Barbs

Just observing outputs is natural if observers are processes in the asynchronous π-calculus, but what if we had a synchronous observer? In Section 2.4.2 will introduce synchronous π-calculi where processes are affected by receipt of their messages, but it is easy to see how to augment the rules generating asynchronous barbs to also detect inputs.

$$\frac{}{x(\vec{v}).\mathsf{P}\downarrow^s_x} \qquad \frac{}{!x(\vec{v}).\mathsf{P}\downarrow^s_x} \qquad \frac{}{\overline{x}\langle\vec{y}\rangle\downarrow^s_x} \qquad \frac{\mathsf{P}\downarrow^s_x}{\mathsf{P}\mid\mathsf{Q}\downarrow^s_x} \qquad \frac{\mathsf{P}\downarrow^s_x \quad x\neq y}{(\nu y)\mathsf{P}\downarrow^s_x}$$

LEMMA 2 *If* $\mathsf{P} \equiv \mathsf{Q}$*, then* $\mathsf{P}\downarrow^s_x$ *implies* $\mathsf{Q}\downarrow^s_x$ *(*$\mathsf{P}\downarrow_x$ *implies* $\mathsf{Q}\downarrow_x$*).*

The next theorem shows that these *synchronous barbs* cannot be recovered from asynchronous barbs by closure under contexts.

THEOREM 3 *If* $\overset{rc}{\approx}_s$ *is the reduction congruence induced by synchronous barbs then* $\overset{rc}{\approx}_s \subsetneq \overset{rc}{\approx}$.

[61] discusses the differences between these relations in more detail.

Other Barbs and Congruences Without Barbs

Of course $\downarrow_x$ and $\downarrow^s_x$ are far from being the only notions of observations we may consider. Fournet and Gonthier consider other choices and discuss how the resulting equivalences relate in the asynchronous π-calculus [38] and in Join calculi [37]. In Chapter 4 we will equip observers with clocks that can tell how much time it takes a process to do computations. This has dramatic influences on the resulting congruence.

This proliferation of barbs calls for a general theory that allows to decide on the right choice of observation predicate, or even suggest barbs for calculi where it is not clear what they might look like (such as Ambient calculi [25]) In [61], Honda and Yoshida propose an important step towards this goal. Their approach is based on two main ideas.

- Instead of requiring barb preservation with all the associated problems of finding the right notion of barb, they suggest to identify a basic set of equations that any reasonable equivalence must contain.
- They introduce the concept of *insensitive* processes and propose to require their equality by definition. As described in (4), processes are insensitive if they cannot possible communicate with other processes. It turns out that the notion of insensitivity is easily defined from the syntax and reduction semantics of processes, without any additional notion of observation.

[61] defines a relation, also called reduction congruence and denoted $\overset{rc}{\approx}$, as the largest consistent congruence that is reduction-closed and identifies all insensitive terms. Identification of insensitive terms is vital to ensure uniqueness of definition. In the asynchronous π-calculus, this definition of reduction congruence coincides with our Definition 3. This sets the pattern for classifying barbs: only if $\overset{rc}{\approx}_B$ coincides with $\overset{rc}{\approx}$, B is an appropriate notion of observation.

Notational Convention

We will often have to construct equivalences $\mathcal{R}$. To simplify this process, we assume $\mathcal{R}$ to be the least symmetric relation on processes containing a set of pairs. This set of pairs will usually be specified as follows.

$$\begin{array}{lcll} \mathsf{P}_1 & \mathcal{R} & \mathsf{Q}_1 & \text{whenever } \phi(\mathsf{P}_1, \mathsf{Q}_1) \\ & \vdots & & \\ \mathsf{P}_n & \mathcal{R} & \mathsf{Q}_n & \text{whenever } \phi(\mathsf{P}_n, \mathsf{Q}_n) \end{array}$$

Here the ϕ_i are binary predicates on processes and only those pair that meet the appropriate predicate should be in in the set of processes that induces $\mathcal{R}$. We omit ϕ_i if it is trivial. We may compress the definition even more by writing things like

$$\left.\begin{array}{l}\mathsf{P}_1\\ \vdots\\ \mathsf{P}_m\end{array}\right\}\ \mathcal{R}\ \left\{\begin{array}{ll}\mathsf{Q}_1 & \\ \vdots & \text{whenever } \phi(\vec{\mathsf{P}},\vec{\mathsf{Q}})\\ \mathsf{Q}_n & \end{array}\right.$$

which stands for

$$\begin{array}{lcll}\mathsf{P}_1 & \mathcal{R} & \mathsf{Q}_1 & \text{whenever } \phi(\mathsf{P}_1,\mathsf{Q}_1)\\ & \vdots & & \\ \mathsf{P}_m & \mathcal{R} & \mathsf{Q}_1 & \text{whenever } \phi(\mathsf{P}_m,\mathsf{Q}_1)\\ \mathsf{P}_1 & \mathcal{R} & \mathsf{Q}_2 & \text{whenever } \phi(\mathsf{P}_1,\mathsf{Q}_2)\\ & \vdots & & \\ \mathsf{P}_m & \mathcal{R} & \mathsf{Q}_2 & \text{whenever } \phi(\mathsf{P}_m,\mathsf{Q}_2)\\ & \vdots & & \\ \mathsf{P}_1 & \mathcal{R} & \mathsf{Q}_n & \text{whenever } \phi(\mathsf{P}_1,\mathsf{Q}_n)\\ & \vdots & & \\ \mathsf{P}_m & \mathcal{R} & \mathsf{Q}_n & \text{whenever } \phi(\mathsf{P}_m,\mathsf{Q}_n).\end{array}$$

2.3.3 Transitional Semantics

In practise, the equivalences considered so far are difficult to use for reasoning. The problem lies with quantification over contexts for ensuring congruency. A closer look at why this quantification is required suggests that it is due to the way barbs are matched: for example, if $\mathsf{P}\downarrow_x$, we are given information about P but not about the process that P evolves into when the observation takes place. This has the effect that the definitions 3 and 4 above cannot directly place constraints on how the continuations of observed processes are to be matched. It is only indirectly, through closure under contexts, that these continuations are related.

Labelled transitions are an alternative to reductions that allow to directly constrain how continuations of observed processes are related. As mentioned in the introduction to this section, labelled transitions are of the form $\mathsf{P}\xrightarrow{l}\mathsf{Q}$ and can be read as "the process P allows an observer to make the observation l and while the observation takes place, P evolves into Q". The observation l can be understood as a *minimal* offering (computation, observation) an environment has to make for P to be able to evolve into Q. Labelled transitions allow alternative characterisations of important equivalences such as reduction congruence (at least in the asynchronous π-calculus) without requiring closure under arbitrary context. Consequently, they often simplify reasoning. Hence labelled transitions can be understood as a tool to obtain interesting results about equivalences defined using reductions.

Equivalences defined using labelled transitions work by matching transitions $\mathsf{P} \xrightarrow{l} \mathsf{P}'$ with sequences $\mathsf{Q} \to \cdots \to \xrightarrow{l} \to \cdots \to \mathsf{Q}'$ and requiring P' and Q' to be equated. As we do not want labelled transitions to be mixed with reductions, we need a distinguished label τ to denote internal interaction that is invisible to observers: if $\mathsf{P} \xrightarrow{\tau} \mathsf{Q}$ then P takes a step in its computation to become Q. Such transitions correspond to reductions in the sense that we certainly want $\mathsf{P} \to \mathsf{Q}$ iff $\mathsf{P} \equiv \xrightarrow{\tau} \equiv \mathsf{Q}$ to hold. It can be shown that for the calculi we are interested in, this is indeed the case.

To define labelled equivalences as sketched in the previous paragraph, we must be able to formally "ignore" τ transitions. The next definition provides a tool to do this.

Definition 7 Let A be a set not containing τ. The function $\hat{s}$ maps strings s over $\{\tau\} \cup A$ to strings over A, removing all occurrences of τ while preserving all other letters and their ordering.

$$\hat{s} = \begin{cases} \langle\rangle & s = \langle\rangle \\ \hat{t} & s = \tau.t \\ a.\hat{t} & s = a.t, a \neq \tau \end{cases}$$

We have now assembled the tools to define bisimilarity, the canonical labelled equivalence. Here and subsequently we omit to explicitly mention the labelled transition system relative to which bisimilarity is defined, hoping that disambiguation is always possible from the context.

Definition 8 A symmetric binary relation $\mathcal{R}$ on processes is a *bisimulation* if $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ and $\mathsf{P} \overset{s}{\twoheadrightarrow} \mathsf{P}'$ imply the existence of a labelled transition sequence $\mathsf{Q} \overset{t}{\twoheadrightarrow} \mathsf{Q}'$ such that $\hat{s} = \hat{t}$ and $(\mathsf{P}', \mathsf{Q}') \in \mathcal{R}$. The union of all bisimulations is itself a bisimulation and called *bisimilarity*. It is often denoted $\approx$.

As in the previous section, we can require more stringent matching of computational steps, which means that τ is not given a special status as an invisible action that can be ignored by observers.

Definition 9 A symmetric binary relation $\mathcal{R}$ on processes is a *strong bisimulation* if $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ and $\mathsf{P} \xrightarrow{a} \mathsf{P}'$ imply the existence of a labelled transition $\mathsf{Q} \xrightarrow{a} \mathsf{Q}'$ such that $(\mathsf{P}', \mathsf{Q}') \in \mathcal{R}$. The union of all bisimulations is itself a strong bisimulation and called *strong bisimilarity*. It is often denoted $\sim$.

The following theorem characterises bisimilarity and simplifies checking processes for being bisimilar.

Theorem 4 $\mathsf{P} \approx \mathsf{Q}$ *iff*

- *whenever* $\mathsf{P} \stackrel{\tau}{\rightarrow} \mathsf{P}'$ *then there is* Q' *such that* $\mathsf{Q} \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} \mathsf{Q}'$ *and* $\mathsf{P}' \approx \mathsf{Q}'$,
- *whenever* $\mathsf{P} \stackrel{l}{\rightarrow} \mathsf{P}'$ *for* $l \neq \tau$ *then there is* Q' *such that* $\mathsf{Q} \underbrace{\stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow}}_{\geq 0} \stackrel{l}{\rightarrow} \underbrace{\stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow}}_{\geq 0} \mathsf{Q}'$ *and* $\mathsf{P}' \approx \mathsf{Q}'$,

and vice versa.

Given a reduction based equivalence, say $\stackrel{r}{\approx}$, the problem is to define labelled transitions such that not only $\rightarrow$ and $\stackrel{\tau}{\rightarrow}$ coincide up to $\equiv$, but also to ensure that $\approx = \stackrel{r}{\approx}$, or at least $\approx \subseteq \stackrel{r}{\approx}$. It turns out that achieving $\approx = \stackrel{r}{\approx}$ is difficult in the asynchronous π-calculus, but $\approx \subseteq \stackrel{r}{\approx}$ is quite doable. In practise, $\approx \subseteq \stackrel{r}{\approx}$ is usually sufficient for reasoning, because $\approx$ misses coincidence with $\stackrel{r}{\approx}$ only just and the processes that are equated by $\stackrel{r}{\approx}$ but not by $\approx$ are often not the ones we are interested in equating.

Trace Equivalences

Bisimulations of various sorts are usually the equivalences of choice for compositional reasoning about concurrent systems but they are not without flaws. Firstly, they are often too discriminating when considering systems as a whole rather than as components to be composed with other components. For example, most bisimulations do not equate $\mathsf{P} \oplus (\mathsf{Q} \oplus \mathsf{R})$ with $(\mathsf{P} \oplus \mathsf{Q}) \oplus \mathsf{R}$ because of their differing branching structure. This seems to be the price we have to pay for congruency and reduction closure, which are indispensable for compositional reasoning. Secondly, because they are defined coinductively, it is often quite tricky to show directly that two processes are *not* bisimilar. Traces [52] help overcome both problems at once. Since they are not sensitive to branching structure, they equate many more processes than bisimulations, in particular the above $\mathsf{P} \oplus (\mathsf{Q} \oplus \mathsf{R})$ and $(\mathsf{P} \oplus \mathsf{Q}) \oplus \mathsf{R}$. It is also mostly straightforward to verify that two processes are not trace-equivalent. It is this latter property that we will use in this text because bisimilarity is a proper subrelation of trace-equivalence: so for showing two processes not to be bisimilar it is enough to establish that they are not trace-equivalent.

DEFINITION 10 The sets of *traces* and *strong traces* of a process P, respectively denoted $\mathsf{tr}(\mathsf{P})$ and $\mathsf{str}(\mathsf{P})$, have this definition:

$$\begin{aligned} \mathsf{str}(\mathsf{P}) &= \{\sigma \mid \exists \mathsf{P}_1 ... \mathsf{P}_n . \mathsf{P} \stackrel{l_1}{\rightarrow} \mathsf{P}_1 \stackrel{l_2}{\rightarrow} \cdots \stackrel{l_n}{\rightarrow} \mathsf{P}_n, \sigma = \langle l_1 ... l_n \rangle\}, \\ \mathsf{tr}(\mathsf{P}) &= \{\hat{\sigma} \mid \sigma \in \mathsf{str}(\mathsf{P})\}. \end{aligned}$$

Two processes P and Q are *trace equivalent*, denoted $\mathsf{P} \approx_{\mathsf{tr}} \mathsf{Q}$ if $\mathsf{tr}(\mathsf{P}) = \mathsf{tr}(\mathsf{Q})$. They are *strongly trace equivalent* if $\mathsf{str}(\mathsf{P}) = \mathsf{str}(\mathsf{Q})$. In the latter case we write $\mathsf{P} \sim_{\mathsf{str}} \mathsf{Q}$.

As you can see, the only difference between the definition of traces and bisimulations is that traces do *not* require relating the continuations of observations while

bisimulations do. The next theorem summarises the relationship between traces and reduction congruence. It is stated for the asynchronous π-calculus, but holds of any reasonable process calculus.

THEOREM 5 ([54]) *In the asynchronous π-calculus: if $\mathsf{P} \overset{rc}{\approx} \mathsf{Q}$ then $\mathsf{tr}(\mathsf{P}) = \mathsf{tr}(\mathsf{Q})$. The reverse implication does not hold.*

We will now present three different labelled transition systems for the asynchronous π-calculus. The first two approximate $\overset{rc}{\approx}$ and $\overset{rc}{\approx}_s$ while the third rephrases the second, but is more useful for practical reasoning.

Asynchronous Transitions

A *labelled transition system* comprises a set S of *states*, a set Σ of *labels* or *actions* and a family $(\overset{l}{\rightarrow})_{l \in \Sigma}$ of transitions which are binary relations on S.

The set of labels for the asynchronous π-calculus is given by the following grammar.

$$l ::= \tau \;\mid\; \overline{x}\langle(\nu\vec{z})\vec{y}\rangle \;\mid\; x(\vec{v})$$

Here x ranges over names and $\vec{x}, \vec{y}, \vec{z}$ and $\vec{v}$ range over finite tuples of names. We abbreviate $\overline{x}\langle(\nu())\vec{y}\rangle$ to $\overline{x}\langle\vec{y}\rangle$, $\overline{x}\langle()\rangle$ to $\overline{x}$ and $x(())$ to x. We will often have to speak about the *free* and *bound names* of a label l. The following equations provide suitable definitions.

$$\begin{array}{lcl@{\qquad}lcl}
\mathsf{fn}(\tau) & = & \emptyset & \mathsf{bn}(\tau) & = & \emptyset \\
\mathsf{fn}(\overline{x}\langle(\nu\vec{z})\vec{y}\rangle) & = & \{x\} \cup (\{\vec{y}\} \setminus \{\vec{z}\}) & \mathsf{bn}(\overline{x}\langle(\nu\vec{z})\vec{y}\rangle) & = & \{\vec{z}\} \\
\mathsf{fn}(x(\vec{v})) & = & \{x\} & \mathsf{bn}(x(\vec{v})) & = & \{\vec{v}\}
\end{array}$$

The rules for asynchronous transitions are given in Figure 2.5. Rules involving the parallel combinator have a symmetric version which we omit here and subsequently. Before explaining the rules in detail, we state two essential facts connecting labelled transitions with their reduction based counterpart.

THEOREM 6 *[75] Transition and reduction semantics coincide up to structural congruency, that is: $\rightarrow \;=\; (\overset{\tau}{\rightarrow} \circ \equiv)$.*

This theorem shows that τ does indeed capture computation in the asynchronous π-calculus. The next one says that $\approx$ soundly approximates reduction congruence.

THEOREM 7 *[75] Bisimilarity approximates reduction congruence: $\approx \subseteq \overset{rc}{\approx}$.*

The (OUT) rule says that an asynchronous observer of $\overline{x}\langle\vec{y}\rangle$, i.e. another process in the asynchronous π-calculus, can observe the emission of $\overline{x}\langle\vec{y}\rangle$, by executing the corresponding input that consumes the message. As a consequence of this interaction, the observed process would evolve into $\mathsf{0}$. The (IN_a) is symmetric: on observer

$$(\textsc{Out}) \quad \frac{}{\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{0}}$$

$$(\textsc{In}_a) \quad \frac{}{\mathsf{0} \xrightarrow{x(\vec{y})}_a \overline{x}\langle\vec{y}\rangle}$$

$$(\textsc{Par}) \quad \frac{\mathsf{P} \xrightarrow{l}_a \mathsf{P}' \qquad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{l}_a \mathsf{P}' \mid \mathsf{Q}}$$

$$(\textsc{Com}) \quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid x(\vec{v}).\mathsf{Q} \xrightarrow{\tau}_a \mathsf{Q}\{\vec{y}/\vec{v}\}}$$

$$(\textsc{Rep}) \quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid\, !x(\vec{v}).\mathsf{Q} \xrightarrow{\tau}_a \mathsf{Q}\{\vec{y}/\vec{v}\} \mid\, !x(\vec{v}).\mathsf{Q}}$$

$$(\textsc{Res}) \quad \frac{\mathsf{P} \xrightarrow{l}_a \mathsf{Q} \qquad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)}{(\nu x)\mathsf{P} \xrightarrow{l}_a (\nu x)\mathsf{Q}}$$

$$(\textsc{Open}) \quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q} \qquad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle}_a \mathsf{Q}}$$

$$(\textsc{Cong}) \quad \frac{\mathsf{P} \equiv \mathsf{P}' \qquad \mathsf{P}' \xrightarrow{l}_a \mathsf{Q}' \qquad \mathsf{Q}' \equiv \mathsf{Q}}{\mathsf{P} \xrightarrow{l}_a \mathsf{Q}}$$

Figure 2.5: The asynchronous transitions for the asynchronous π-calculus.

can *always* send an output to an arbitrary process even if there is no receiving input, because, as the observer is asynchronous, it cannot detect the consumption of its output by the process under observation. In order to cater for the possibility that the sent message is not (immediately) consumed, we simply add it to the observed process for potential latter consumption (it is enough to have $\mathsf{0}$ in this rule because we can run arbitrary processes in parallel with $\mathsf{0}$ without changing observations by the (Par) rule below). The transition, $\mathsf{P} \xrightarrow{x(\vec{y})} \mathsf{Q}$ should be understood as saying "an observer of P can send a message $\overline{x}\langle\vec{y}\rangle$ to P. If this happens, P evolves into Q". Despite this innocent interpretation, (IN_a) is controversial. The problem is that process theorists raised in the CCS tradition interpret transitions $\mathsf{P} \xrightarrow{l} \mathsf{Q}$ as "P can do l while evolving into Q". Under this interpretation, ($\textsc{In}_a$) posits $\mathsf{0}$ to be able to do an input and evolve into an output. This would be plainly nonsense, as $\mathsf{0}$ is the prototypical inactive process. It is possible to produce a labelled semantics of the asynchronous π-calculus that allows to approximate $\overset{rc}{\approx}$ by bisimilarity without violating this conventional interpretation of labelled transitions [7]. The price to pay is that the definition of bisimulation must be modified considerably in ways that are inappropriate for other semantic scenarios, for example for synchronous observers. The modifications essentially mimic the ability to add arbitrary messages to the process under observation. The (Par) rule says that observations are not affected by parallel composition with other processes. (Com) and (Rep) allow to infer that

the observed process computes internally, but they do not leak specifics about this computation.

So far we have only talked about outputs of the form $\overline{x}\langle\vec{y}\rangle$, so what is the point of having the more general $\overline{x}\langle(\nu\vec{y})\vec{z}\rangle$? And how is observation of scope extension accounted for? Consider $(\nu y)(\overline{x}\langle yz\rangle \mid \mathsf{P})$: if $x \neq y$, then an observer of the form $x(vw).\mathsf{Q}$ can detect the bound output on x by enacting the dual input. This is reflected in the label $\overline{x}\langle(\nu\vec{y})\vec{z}\rangle$ which lists all the names $\vec{y}$ that can only be received by scope extension. So clearly $x \notin \{\vec{y}\}$. In addition one can easily show $\{\vec{y}\} \subseteq \vec{z}$. But what is the continuation of the process under observation? Clearly

$$(\nu y)(\overline{x}\langle yz\rangle \mid \mathsf{P}) \mid x(vw).\mathsf{Q} \rightarrow (\nu y)(\mathsf{P} \mid \mathsf{Q}\{yz/vw\})$$

is a different computation from

$$\overline{x}\langle yz\rangle \mid \mathsf{P} \mid x(vw).\mathsf{Q} \rightarrow \mathsf{P} \mid \mathsf{Q}\{yz/vw\}$$

because $\mathsf{P} \mid \mathsf{Q}\{yz/vw\}$ can be interfered with from the outside by sending messages on y. This is not possible for $(\nu y)(\mathsf{P} \mid \mathsf{Q}\{yz/vw\})$. (Res) formalises that observations are not affected by hiding of a name x, as long as x does not occur in the observation, but if a bound output is observed the restriction is taken off from the observed process. Finally, (Cong) closes observations under the structural congruence.

By $\approx_a$ we denote the bisimilarity induced by asynchronous transitions. The next result shows that $\approx_a$ is semantically sound, but does not capture $\overset{rc}{\approx}$.

Theorem 8 *([54])* $\approx_a \subset \overset{rc}{\approx}$.

[61] discusses this matter further and argues that the gap between $\approx_a$ and $\overset{rc}{\approx}$ is rather small.

Synchronous Transitions (1)

The labelled transitions presented in the previous section essentially formalise asynchronous observers. It is also possible to give labelled semantics corresponding to synchronous observers, Figure 2.6 gives one way of doing this. The only difference from Figure 2.5 is in the rule for input. In its synchronous form it essentially says that an observer of $x(\vec{y}).\mathsf{P}$ can observe the execution of this input, by doing a matching output on x. This observation is of course only possible if doing an output has an effect on the observer, in other words, if the observer is a synchronous process. It is not possible in the asynchronous π-calculus. In §2.4.2 we modify the syntax slightly to obtain the synchronous π-calculus. This boils down to allowing non-trivial output continuations.

$$\text{(Out)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle} \mathsf{0}}$$

$$\text{(In)}\quad \frac{}{x(\vec{v}).\mathsf{P} \xrightarrow{x(\vec{z})} \mathsf{P}\{\vec{z}/\vec{v}\}}$$

$$\text{(Par)}\quad \frac{\mathsf{P} \xrightarrow{l} \mathsf{P}' \quad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{l} \mathsf{P}' \mid \mathsf{Q}}$$

$$\text{(Com)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid x(\vec{v}).\mathsf{Q} \xrightarrow{\tau} \mathsf{Q}\{\vec{y}/\vec{v}\}}$$

$$\text{(Rep)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid !x(\vec{v}).\mathsf{Q} \xrightarrow{\tau} \mathsf{Q}\{\vec{y}/\vec{v}\} \mid !x(\vec{v}).\mathsf{Q}}$$

$$\text{(Res)}\quad \frac{\mathsf{P} \xrightarrow{l} \mathsf{Q} \quad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)}{(\nu x)\mathsf{P} \xrightarrow{l} (\nu x)\mathsf{Q}}$$

$$\text{(Open)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{Q} \quad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle} \mathsf{Q}}$$

$$\text{(Cong)}\quad \frac{\mathsf{P} \equiv \mathsf{P}' \quad \mathsf{P}' \xrightarrow{l} \mathsf{Q}' \quad \mathsf{Q}' \equiv \mathsf{Q}}{\mathsf{P} \xrightarrow{l} \mathsf{Q}}$$

Figure 2.6: The synchronous transitions for the asynchronous π-calculus.

The actual computations of the asynchronous π-calculus are not affected by having synchronous rather than asynchronous observations, as the next theorem shows.

THEOREM 9 *[75] Transition and reduction semantics coincide, that is:* $\rightarrow = \stackrel{\tau}{\rightarrow} \equiv$.

The following theorem shows that just as asynchronous observations allow sound approximation of the reduction congruence, so do synchronous observers. But as an approximation of reduction congruence, $\approx_s$, the bisimilarity induced by synchronous transitions, is worse than $\approx_a$.

THEOREM 10 *([54])* $\approx_s \subset \approx_a \subset \stackrel{rc}{\approx}$.

Nevertheless, for many purposes, $\approx_s$ is the most convenient choice for reasoning about $\stackrel{rc}{\approx}$.

Synchronous Transitions (2)

The two labelled semantics presented so far, while appealing in their simplicity and uniformity, have one basic shortcoming in that they require explicit closure under $\equiv$. This is essential to allow inference of interaction between processes that are more complicated than those in the (Com) rule. Figure 2.7 presents labelled semantics for synchronous observers that overcomes this problem. This type of semantics is

$$\text{(Out)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle} \mathsf{0}}$$

$$\text{(In)}\quad \frac{}{x(\vec{v}).\mathsf{P} \xrightarrow{x(\vec{z})} \mathsf{P}\{\vec{z}/\vec{v}\}}$$

$$\text{(Par)}\quad \frac{\mathsf{P} \xrightarrow{l} \mathsf{P}' \quad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{l} \mathsf{P}' \mid \mathsf{Q}}$$

$$\text{(Com)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{x(\vec{z})} \mathsf{Q}' \quad \{\vec{y}\} \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\overline{x}\langle\vec{y}\rangle \mid x(\vec{v}).\mathsf{Q} \xrightarrow{\tau} \mathsf{Q}\{\vec{y}/\vec{v}\}}$$

$$\text{(Rep)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{x(\vec{z})} \mathsf{Q}' \quad \{\vec{y}\} \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\overline{x}\langle\vec{y}\rangle \mid\, !x(\vec{v}).\mathsf{Q} \xrightarrow{\tau} \mathsf{Q}\{\vec{y}/\vec{v}\} \mid\, !x(\vec{v}).\mathsf{Q}}$$

$$\text{(Res)}\quad \frac{\mathsf{P} \xrightarrow{l} \mathsf{Q} \quad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)}{(\nu x)\mathsf{P} \xrightarrow{l} (\nu x)\mathsf{Q}}$$

$$\text{(Open)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{Q} \quad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle} \mathsf{Q}}$$

$$\text{(Alpha)}\quad \frac{\mathsf{P}' \xrightarrow{l} \mathsf{Q} \quad \mathsf{P} \equiv_\alpha \mathsf{P}'}{\mathsf{P} \xrightarrow{l} \mathsf{Q}}$$

Figure 2.7: The synchronous transitions for the asynchronous π-calculus presented without a need for closure under $\equiv$.

usually the most convenient for practical reasoning. Semantics for asynchronous observers that do not require explicit closure under $\equiv$ are not currently known.

The key novelty of the standard synchronous semantics is that (Com) is deduced from dual observations rather than directly from the syntax. If we can observe $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{P}'$ but also a dual output $\mathsf{Q} \xrightarrow{x(\vec{z})} \mathsf{Q}'$, then an interaction $\mathsf{P} \mid \mathsf{Q} \xrightarrow{\tau} (\nu\vec{y})(\mathsf{P}' \mid \mathsf{Q}')$ can be inferred. This does not require the interacting input and output to be in syntactic proximity. Hence we do not need $\equiv$ to rearrange syntax.

The two synchronous semantics for the asynchronous π-calculus, presented in Figures 2.6 and 2.7 coincide.

Theorem 11 *Transitions $\xrightarrow{l}$ derivable in Figure 2.6 coincide (up to $\equiv$) with those derivable in Figure 2.7. The two induced notions of bisimulation and hence bisimilarity also coincide.*

2.3.4 Examples

Let's consider some examples of processes and their behaviour. If P and Q are two processes,

$$\mathsf{P} \oplus \mathsf{Q} = (\nu x)(\overline{x} \mid x.\mathsf{P} \mid x.\mathsf{Q})$$

(where x is a fresh name) is called the *internal choice* between P and Q. It selects P or Q, but the outside cannot influence which. Selection in this context mean that $\mathsf{P} \oplus \mathsf{Q}$ has exactly two immediate reductions (up to $\equiv$):

$$\mathsf{P} \oplus \mathsf{Q} \to \mathsf{P} \mid (\nu x)x.\mathsf{Q} \qquad \text{or} \qquad \mathsf{P} \oplus \mathsf{Q} \to \mathsf{Q} \mid (\nu x)x.\mathsf{P}.$$

Every reasonable equivalence (in particular, all we have presented in this chapter, apart from the structural congruence $\equiv$ which is a tool to conveniently define reduction semantics, not a proper process equivalence) equates P with $\mathsf{P} \mid (\nu x)x.\mathsf{Q}$ and Q with $\mathsf{Q} \mid (\nu x)x.\mathsf{P}$.

Although most π-calculi do not have basic data types such as booleans or natural numbers, it it easy to define them as interacting agents in ways that resemble how one solves the same problem for λ-calculi [11]. As an example, consider the implementation of booleans.

$$\mathsf{true}_x = x(yz).\overline{y} \qquad\qquad \mathsf{false}_x = x(yz).\overline{z}$$

Both processes wait to receive two names and then choose to send an empty message along one of them. This choice represents a truth value and is intended to be consumed by a conditional:

$$\mathsf{if}(\mathsf{B},\mathsf{P},\mathsf{Q}) = (\nu xyz)(y.\mathsf{P} \mid z.\mathsf{Q} \mid \overline{x}\langle yz\rangle) \mid \mathsf{B}.$$

One peculiar feature of this implementation of booleans is that it is destructive in the sense that once a truth value has been communicated, the truth values disappear as processes. It is easy to rectify this by setting $\mathsf{true}_x = !x(yz).\overline{y}$ and similarly for false_x. For practical implementations one would not implement basic values as interacting processes, because it would be cumbersome, but rather extend the calculus with primitives for basic data types, again, much like in λ-calculi.

Let x and y be two names. A *forwarder* relays to y a message destined for an interaction point x.

$$\mathsf{fw}_{xy} = !x(\vec{v}).\overline{y}\langle\vec{v}\rangle$$

Then

$$\mathsf{fw}_{xy} \mid \overline{x}\langle abc\rangle \to \mathsf{fw}_{xy} \mid \overline{y}\langle abc\rangle$$

A particularly interesting forwarder is the *identity receptor*

$$\mathsf{id}_x = \mathsf{fw}_{xx}$$

because it separates two types of process equivalences: those that equate id_x with $\mathsf{0}$ and those that don't. The former are sometimes considered synchronous while the latter are asynchronous. The reason for this terminology is that id_x can do an input while $\mathsf{0}$ cannot, but that input can only be detected by synchronous observers.

Honda bases an extensive study of bisimilarities for asynchronous π-calculi on this peculiarity of identity receptors [54].

An *equator* is made up of two forwarders

$$\mathsf{eq}_{xy} = \mathsf{fw}_{xy} \,|\, \mathsf{fw}_{yx}.$$

It forwards messages from x to y and vice versa. It can easily lead to non-termination.

$$\mathsf{eq}_{xy} \,|\, \overline{x}\langle abc\rangle \rightarrow \mathsf{eq}_{xy} \,|\, \overline{y}\langle abc\rangle \rightarrow \mathsf{eq}_{xy} \,|\, \overline{x}\langle abc\rangle \rightarrow \mathsf{eq}_{xy} \,|\, \overline{y}\langle abc\rangle \rightarrow \ldots$$

A process that cannot do anything apart from infinite looping is

$$\Omega = (\nu x)(\mathsf{id}_x \,|\, \overline{x}).$$

2.4 Syntactic Variants

The asynchronous π-calculus is probably the simplest example of a π-calculus but it is hardly canonical. Many syntactic and semantic variants are possible and this lack of canonicity is sometimes taken as an indication that our understanding of name passing is still incomplete [42]. In this section we present some important variants of π-calculi and sketch how they relate. The variants have been chosen because of their historical importance, for didactic purposes or because they will be used in later chapters. An interesting question is if the extensions we propose later to model DS will coexist peacefully with these variants, in the same way they do with the asynchronous π-calculus.

With the proliferation of π-calculus variants the problem of their relative expressiveness becomes pertinent. It is easy to show that all variants discussed here can encode Turing Machines, so, putting faith in the Church-Turing Thesis, despite the reservations to be discussed in §4.10, it is reasonable to assume that they are all equally expressive in the weak sense that they can all somehow "simulate" each other. This formulation of expressiveness is unsatisfactory for two reasons: firstly, the more disparate the computational formalisms under comparison, the less clear it is just what counts as simulation. Secondly, the mere fact that some sort of simulation is possible does not clarify how easy, effective or natural that simulation may or can be. Practical programming experience often shows that certain types of problems are much more readily solved in one programming language than in another. The Church-Turing Thesis glosses over such expressivity differences.

Current research on programming language expressivity deals with both issues with the help of *encodings*. An encoding maps programs P in the source language to target programs $[\![\mathsf{P}]\!]$. The more constraints $[\![\cdot]\!]$ meets, the more expressive the target language is considered to be. Alas, as yet no fully convincing set of constraints

has been agreed on, and it may even be the case that no single set of constraints will lead to an unanimous classification of language expressivity. Currently, some requirements are generally assumed to be indispensable. For example, we expect encodings to be *sound*:

$$[\![\mathsf{P}]\!] \approx_t [\![\mathsf{Q}]\!] \qquad \text{implies} \qquad \mathsf{P} \approx_s \mathsf{Q}$$

where $\approx_t$ is an equivalence for target language and $\approx_s$ is likewise for the source language. For an encoding to be useful for reasoning, *full abstraction* is probably a minimal requirement:

$$[\![\mathsf{P}]\!] \approx_t [\![\mathsf{Q}]\!] \qquad \text{if and only if} \qquad \mathsf{P} \approx_s \mathsf{Q}.$$

If the operations of the source and target language are sufficiently similar, one usually requires preservation of some operators by the encoding: if, for example, source and target are both process calculi, we would expect

$$[\![\mathsf{P} \mid \mathsf{Q}]\!] = [\![\mathsf{P}]\!] \mid [\![\mathsf{Q}]\!]$$

to hold. Preservation of an operator f is referred to as *compositionality with respect to* f. Preservation of $\mid$ is simply *compositionality*. Surprisingly, it is almost always possible to find compositional and fully abstract encodings but they are computationally meaningless [12]. It is not even clear exactly what makes an encoding computationally meaningful, but often *computational correspondence* is taken to be sufficient. Computational correspondence essentially means that the computational steps are decomposed *in a uniform manner* into computational steps of the target calculus. It is important to realise that soundness, full abstraction, compositionality and the like are all defined with respect to two equivalences, one for the source and one for the target calculus. Some such property might hold for $[\![\cdot]\!]$ with respect to some equivalences but not for others.

Unfortunately, not all the π-calculus variants we shall talk about are known to have fully abstract compositional encodings into the asynchronous π-calculus which exhibit computational correspondence, although some do. We believe that at least in the case of π-calculi with timers, like that to be introduced in Chapter 4, such encodings are in principle impossible. In addition, some, if not most, of the known encodings for the π-calculi variants we are going to present in the rest of this chapter, will not work in the presence of the extensions for DS to be introduced subsequently.

In summary, expressiveness of computational formalisms in general and π-calculi in particular is badly understood and deserves closer scrutiny.

2.4.1 Recursion and Other Forms of Replication

For a computational formalism to be universal in the sense of the Church-Turning Thesis, the ability to express infinite behaviour is indispensable. To be pragmatical-

ly viable, the syntax of a formalism must be finite. In the asynchronous-π-calculus, (lazy) replication bridges the gap between finite syntax and infinite behaviour. Historically, lazy replication in π-calculi was probably conceived in analogy with the !-modality of linear logic [41, 75] but has a close conceptual match in process forking, as customarily found in many operating systems [10]. Nevertheless, it is sometimes more convenient to specify processes using recursive equations, for example

$$\mathsf{P}\langle xy\rangle = \overline{x}\langle y\rangle \,|\, \mathsf{Q}\langle z\rangle \qquad \mathsf{Q}\langle x\rangle = x(v).\mathsf{R}\langle v\rangle \qquad \mathsf{R}\langle x\rangle = \ldots$$

It is straightforward to modify the asynchronous-π-calculus to replace lazy replication by recursive equations:

P	$::=$	$x(\vec{y}).\mathsf{P}$	input prefix
	$\mid$	$\overline{x}\langle\vec{y}\rangle$	output
	$\mid$	$\mathsf{P}\,\mid\mathsf{Q}$	parallel composition
	$\mid$	$(\nu x)\mathsf{P}$	hiding
	$\mid$	$X\langle\vec{y}\rangle$	instantiated agent variable
	$\mid$	$\mathsf{0}$	inaction

A *recursive equation* is then simply a pair $X\langle\vec{y}\rangle = \mathsf{P}$ such that $\mathsf{fn}(\mathsf{P}) = \{\vec{y}\}$. It is usually a good idea to restrict recursive definitions to sets of equations that are decidable and to ensure that each agent variable has exactly one defining equation.

The structural congruence is adjusted by removing all axioms mentioning ! and replacing them by the rule below.

$$\frac{\mathsf{fn}(\mathsf{P}) \subseteq \{\vec{v}\} \qquad X\langle\vec{v}\rangle = \mathsf{P}}{X\langle\vec{y}\rangle \equiv \mathsf{P}\{\vec{y}/\vec{v}\}}$$

The corresponding labelled transition systems is obtained from the one in Figures 2.5, 2.6 and 2.7 by removing (Rep) and replacing it by

$$(\textsc{Rec}) \quad \frac{\mathsf{P}\{\vec{y}/\vec{v}\} \stackrel{l}{\rightarrow} \mathsf{Q} \qquad X\langle\vec{v}\rangle = \mathsf{P}}{X\langle\vec{y}\rangle \stackrel{l}{\rightarrow} \mathsf{Q}}$$

It is straightforward to encode lazy replication using recursion in a fully abstract, compositional and computationally corresponding manner. A third alternative are fixpoint operators, well known from λ-calculi [48], which effectively incorporate recursive equations into the syntax of a computational calculus [57, 59]. Conversely, if a process is given by a *finite* set of defining equations, a fully abstract, compositional and computationally corresponding encoding can easily be given [75]. Although we have been unable to confirm this in the literature, we believe that this result holds even for arbitrary *recursive* sets of defining equations.

Of course the restriction to lazy replication is unnecessary. We could allow other forms such as $!\overline{x}\langle\vec{y}\rangle$, or even $!\mathsf{P}$ without placing constraints on the shape of P. If

we'd do the latter, (Rep) would have to be replaced by an additional axiom for $\equiv$:

$$!\mathsf{P} \equiv \mathsf{P} \mid !\mathsf{P}$$

that ensures $!\mathsf{P}$ really acts like the parallel composition of infinitely many copies of P (one could also add $!!\mathsf{P} \equiv !\mathsf{P}$ or $!\mathsf{0} \equiv \mathsf{0}$, but those are not required to infer substantial computation). Both forms of replication are convincingly encodable into each other. Our reason for preferring lazy replication are threefold. Firstly, $!x(\vec{v}).\mathsf{P}$ is the only way we use replication in applications. Secondly, unrestricted replication does not sit well with truly concurrent semantics as it would allow infinitely many interactions to happen at the same time. While that might be acceptable as a limit case, it is more realistic to just have finitely many interactions occurring in one reduction step. Finally, unrestricted replication also causes problems when timers are added to π-calculi, as we do in Chapter 4, where we elaborate on the issue.

It is also possible to combine replication and definition by recursive equations, as we shall do in later chapters. The various rules coexist harmoniously.

2.4.2 Synchrony

We have focused on the asynchronous π-calculus because it is simpler than most alternatives and has nicer equational properties: in particular, many prima facie different equivalences coincide, cf. Theorem 2. This is a fragile property. It fails to hold if the calculus is extended, for example, with a synchronous output operation.

Despite the advantages of asynchrony, the original π-calculus of Milner, Parrow and Walker was synchronous [78]. Synchrony boils down to allowing sequentialisation not only with input but also with output. Hence processes in the synchronous π-calculus are generated by the following grammar.

P	$::=$	$x(\vec{y}).\mathsf{P}$	input prefix
	$\mid$	$!x(\vec{y}).\mathsf{P}$	lazy replication
	$\mid$	$\overline{x}\langle\vec{y}\rangle.\mathsf{P}$	output prefix
	$\mid$	$\mathsf{P} \mid \mathsf{Q}$	parallel composition
	$\mid$	$(\nu x)\mathsf{P}$	hiding
	$\mid$	$\mathsf{0}$	inaction

In the reduction semantics, (Com) and (Rep) need to be modified.

$$(\textsc{Com}) \quad \frac{}{x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle.\mathsf{Q} \;\rightarrow\; \mathsf{P}\{\vec{y}/\vec{v}\} \mid \mathsf{Q}}$$

$$(\textsc{Rep}) \quad \frac{}{!x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle.\mathsf{Q} \rightarrow !x(\vec{v}).\mathsf{P} \mid \mathsf{P}\{\vec{y}/\vec{v}\} \mid \mathsf{Q}}$$

The axioms of the structural congruence remain unchanged. The labelled transitions also only need one modification, here shown for the synchronous variants.

$$(\textsc{Out}) \quad \frac{}{\overline{x}\langle\vec{y}\rangle.\mathsf{P} \xrightarrow{\overline{x}\langle\vec{y}\rangle} \mathsf{P}}$$

Now the synchronous barbs, introduced in §2.3.2 make more sense: they correspond to observing processes being synchronous.

As to expressivity, it is straightforward to embed the asynchronous π-calculus into the synchronous π-calculus. The reverse direction is more involved: the protocol to simulate synchrony is simple enough. Each output $\overline{x}\langle\vec{y}\rangle.\mathsf{P}$ is replaced by an asynchronous output with explicit acknowledgement via a private channel.

$$(\nu a)(\overline{x}\langle\vec{y}a\rangle \mid a.[\![\mathsf{P}]\!])$$

Correspondingly, inputs $x(\vec{v}).\mathsf{P}$ are replaced by

$$x(\vec{v}a).(\overline{a} \mid [\![\mathsf{P}]\!]).$$

This transforms the sequentialisation of the output prefix into a scope extending interaction and input prefix sequentialisation.

As usual, this encoding is sound but not fully abstract. It seems straightforward to adapt the *linear types* of [124] to identify an appropriate set of processes in the synchronous π-calculus to make that makes the encoding fully abstract. Nevertheless, at the time of writing, no fully abstract encoding has been published.

What are the advantages of synchrony over asynchrony other than that it allows to sometimes omit explicit acknowledgements? Probably not many, apart from making the calculus more symmetric. On the other hand, implementations of synchrony, in particular of mixed synchrony on a given channel, for example $x(v).\mathsf{P} \mid \overline{x}\langle y\rangle.\mathsf{Q}$, is detrimental to efficiency [69, 113].

2.4.3 Monadicity

The calculi described so far communicate tuples of names:

$$\overline{x}\langle y_1 ... y_n\rangle \mid x(v_1 ... v_n).\mathsf{P} \to \mathsf{P}\{y_1/v_1 ... y_n/v_n\}.$$

Monadic name-passing is a special case.

$$\overline{x}\langle y\rangle \mid x(v).\mathsf{P} \to \mathsf{P}\{y/v\}$$

Do we loose expressivity when restriction to monadic name passing? Intuitively no, because instead of sending n names at once, we may just as well send them in n separate interactions, possibly along a private channel, to avoid interference. Yoshida [123] as well as Walker and Quaglia [93] have confirmed this hunch with fully abstract encodings. Interestingly, full abstraction fails for their encodings if the target calculus is not typed: decomposition of polyadicity into monadicity changes the granularity of interaction and allows undue interference.

2.4.4 Sums

Sums explicitly introduce *non-determinism* into computations. Of course even without sums, the calculi introduced so far exhibit some form of non-determinism, just consider the process $\overline{x}\,|\,x.\mathsf{P}\,|\,\overline{y}\,|\,y.\mathsf{Q}$. But they lack a dedicated operator for non-determinism. Sums allows processes to *choose* between different continuations. This choice can be understood in several ways. A useful classification is that between internal and external choice.

A process P has *internal choice* if it has at least two distinct continuations Q and R such that $\mathsf{P} \to \mathsf{Q}$ and $\mathsf{P} \to \mathsf{R}$ and the environment has no way to influence which one will be chosen. An example of internal choice is the sum $\mathsf{P} \oplus \mathsf{Q}$, introduced in Section 2.3.4. A process exhibits *external choice* if it offers the choice between several distinct continuations to the environment which will inform P of its choice by way of an interaction.

The original π-calculus [78] is curious if viewed with the aforementioned distinction in mind in that it has a sum that combines internal and external choice. This is most easily elaborated once its syntax and semantics have been presented.

P	$::=$	$x(\vec{y}).\mathsf{P}$	input prefix
	$\mid$	$!x(\vec{y}).\mathsf{P}$	lazy replication
	$\mid$	$\overline{x}\langle\vec{y}\rangle$	output
	$\mid$	$\mathsf{P}\,\|\,\mathsf{Q}$	parallel composition
	$\mid$	$\mathsf{P}+\mathsf{Q}$	binary sum
	$\mid$	$(\nu x)\mathsf{P}$	hiding
	$\mid$	$\mathsf{0}$	inaction

The structural congruence adds axioms to make $+$ monoidal.

- $\mathsf{P}+(\mathsf{Q}+\mathsf{R}) \equiv (\mathsf{P}+\mathsf{Q})+\mathsf{R}$,
- $\mathsf{P}+\mathsf{Q} \equiv \mathsf{Q}+\mathsf{P}$,
- $\mathsf{P}+\mathsf{0} \equiv \mathsf{P}$.

The reduction semantics needs the following modification of its (Com) and (Rep) rules.

$$(\textsc{Com}) \quad \frac{}{(\cdots + \overline{x}\langle\vec{y}\rangle) \mid (x(\vec{v}).\mathsf{P} + \ldots) \to \mathsf{P}\{\vec{y}/\vec{v}\}}$$

$$(\textsc{Rep}) \quad \frac{}{(\cdots + \overline{x}\langle\vec{y}\rangle) \mid (!x(\vec{v}).\mathsf{P} + \ldots) \to !x(\vec{y}).\mathsf{P} \mid \mathsf{P}\{\vec{y}/\vec{v}\}}$$

The transition semantics require two additional and symmetric rules

$$(\textsc{Sum}) \quad \frac{\mathsf{P} \xrightarrow{l} \mathsf{P}'}{\mathsf{P}+\mathsf{Q} \xrightarrow{l} \mathsf{P}'}$$

(omitting, as usual, the symmetric counterpart and asynchronous transitions). External choice between P and Q can now easily be defined.

$$x.\mathsf{P} + y.\mathsf{Q}$$

If a context wants to force this process to become P, it sends an empty message along x and similarly for Q. Internal choice between P and Q, a la $\mathsf{P} \oplus \mathsf{Q}$, can be encoded using $+$ by setting

$$(\nu x)(\overline{x} \,|\, x.\mathsf{P}) + (\nu x)(\overline{x} \,|\, x.\mathsf{Q}).$$

The only thing this process can do is choose to become P or Q, modulo some uninteresting garbage collection.

We can also to combine the two:

$$(\nu x)(\overline{x} \,|\, x.\mathsf{P}) + y.\mathsf{Q} + z.\mathsf{R}.$$

With this process, the environment can choose between Q and R, provided the process itself has not chosen to evolve into P and vice versa. This is mixed choice. In general, a sum term allows *mixed choice* if one of the immediate subterms of $+$ is an input while the other is an output, as for example in $x + \overline{y}$.

Regarding expressivity, $+$, like synchronous output, is a powerful operation that distinguishes several of the equivalences that coincide in the asynchronous π-calculus. Palamidessi [88] has established the impossibility of encoding mixed choice into π-calculus fragments without mixed choice, provided we'd like the encoding to be fully abstract and compositional in addition to some other natural restrictions. On the other hand, *input guarded choice*, where all sums are of the form $x(\vec{v}).\mathsf{P} + y(\vec{w}).\mathsf{Q}$ can be eliminated in the asynchronous π-calculus [84, 85].

While unconstrained forms of summation are problematic in several respects, they are natural for specifications of process behaviour and π-calculi emerged out of an automata theoretic tradition where specifications were important. We will not use unrestricted sums further in this text.

The more constrained external choice is useful in many applications, so much so in fact, that we will augment the calculus with primitives allowing concise expression the availability of alternatives and their external selection. We shall now describe these primitives in the context of the asynchronous π-calculus. The augmented

syntax is given by the following grammar.

$$
\begin{array}{rcll}
\mathsf{P} & ::= & x(\vec{y}).\mathsf{P} & \text{input} \\
& | & !x(\vec{y}).\mathsf{P} & \text{lazy replication} \\
& | & \overline{x}\langle\vec{y}\rangle & \text{output} \\
& | & x[(\vec{y}).\mathsf{P}\ \&\ (\vec{z}).\mathsf{Q}] & \text{branching input} \\
& | & !x[(\vec{y}).\mathsf{P}\ \&\ (\vec{z}).\mathsf{Q}] & \text{lazy branching replication} \\
& | & \overline{x}\mathsf{left}\langle\vec{y}\rangle & \text{left selection} \\
& | & \overline{x}\mathsf{right}\langle\vec{y}\rangle & \text{right selection} \\
& | & \mathsf{P}\,|\,\mathsf{Q} & \text{parallel composition} \\
& | & (\nu x)\mathsf{P} & \text{hiding} \\
& | & \mathsf{0} & \text{inaction}
\end{array}
$$

A term $x[(\vec{y}).\mathsf{P}\ \&\ (\vec{z}).\mathsf{Q}]$ offers to the environment the choice between acting like $x(\vec{y}).\mathsf{P}$ and $x(\vec{z}).\mathsf{Q}$ except that the selection of the choice and the interaction at x happen in one interaction. The arity of $\vec{y}$ and $\vec{z}$ need not coincide. The corresponding selecting outputs are $\overline{x}\mathsf{left}\langle\vec{y}\rangle$ and $\overline{x}\mathsf{right}\langle\vec{y}\rangle$. Their interplay is probably most easily grasped by considering the corresponding rules that need to be added to the usual reduction rules.

$$(\textsc{Com}_l)\quad \frac{}{x[(\vec{v}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{Q}]\ |\ \overline{x}\mathsf{left}\langle\vec{y}\rangle \to \mathsf{P}\{\vec{y}/\vec{v}\}}$$

$$(\textsc{Rep}_l)\quad \frac{}{!x[(\vec{v}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{Q}]\ |\ \overline{x}\mathsf{left}\langle\vec{y}\rangle \to !x[(\vec{v}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{Q}]\ |\ \mathsf{P}\{\vec{y}/\vec{v}\}}$$

(As usual, we omit the symmetric rules.) Free and bound names are computed as expected, we just present one clause of the definition of $\mathsf{fn}(\cdot)$.

$$\mathsf{fn}(x[(\vec{y}).\mathsf{P}\ \&\ (\vec{z}).\mathsf{Q}]) = (\mathsf{fn}(\mathsf{P}) \setminus \{\vec{y}\}) \cup (\mathsf{fn}(\mathsf{Q}) \setminus \{\vec{z}\}) \cup \{x\}$$

In order to have labelled transitions we need to extend the set of labels given in 2.3.3 as follows.

$$l ::= \ldots \ |\ \overline{x}\mathsf{left}\langle(\nu\vec{z})\vec{y}\rangle \ |\ x\mathsf{left}(\vec{v}) \ |\ \overline{x}\mathsf{right}\langle(\nu\vec{z})\vec{y}\rangle \ |\ x\mathsf{right}(\vec{v})$$

The asynchronous labelled transition system needs additional rules for branching output, branching input and additional communication and opening rules. We only

present those for the left branch.

$$(\textsc{Out}_l)\quad \frac{}{\overline{x}\mathsf{left}\langle\vec{y}\rangle \xrightarrow{\overline{x}\mathsf{left}\langle\vec{y}\rangle}_a \mathsf{0}}$$

$$(\textsc{In}_l)\quad \frac{}{x[(\vec{v}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{Q}] \xrightarrow{x\mathsf{left}(\vec{y})}_a \mathsf{P}\{\vec{y}/\vec{v}\}}$$

$$(\textsc{Rep}_l)\quad \frac{}{!x[(\vec{v}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{Q}] \xrightarrow{x\mathsf{left}(\vec{y})}_a \mathsf{P}\{\vec{y}/\vec{v}\} \mid\ !x[(\vec{v}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{Q}]}$$

$$(\textsc{Com})\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\mathsf{left}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{P}' \qquad \mathsf{Q} \xrightarrow{x\mathsf{left}(\vec{z})}_a \mathsf{Q}' \qquad \{\vec{y}\} \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{\tau}_a (\nu\vec{y})(\mathsf{P}' \mid \mathsf{Q}')}$$

$$(\textsc{Open})\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\mathsf{left}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q} \qquad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{P} \xrightarrow{\overline{x}\mathsf{left}\langle(\nu\vec{y},v)\vec{z}\rangle}_a \mathsf{Q}}$$

In general, external choice need not be restricted to the provision of binary alternatives. In fact, to model basic data types such as natural numbers, infinite branching is convenient [16]. For example a process that represents the number 7 located at x could be implemented as

$$!x(r).\overline{r}\mathtt{inj}_7\langle\rangle$$

and the corresponding receiver of such natural numbers could be

$$x[\&_{n\in\mathbb{N}}().\mathsf{P}_n]$$

Binary branching can easily be reduced to asynchronous name passing.

$$\begin{aligned}
[\![\overline{x}\mathsf{left}\langle\vec{y}\rangle]\!] &= (\nu z)(\,\overline{x}\langle z\rangle \mid z(x_l, x_r).\overline{x_l}\langle\vec{y}\rangle\,)\\
[\![\overline{x}\mathsf{right}\langle\vec{y}\rangle]\!] &= (\nu z)(\,\overline{x}\langle z\rangle \mid z(x_l, x_r).\overline{x_r}\langle\vec{y}\rangle\,)\\
[\![x[(\vec{y}).\mathsf{P}\ \&\ (\vec{z}).\mathsf{Q}]]\!] &= x(z).(\nu x_l)(\nu x_r)(\,\overline{z}\langle x_l, x_r\rangle \mid x_l(\vec{y}).[\![\mathsf{P}]\!] \mid x_r(\vec{z}).[\![\mathsf{Q}]\!]\,)
\end{aligned}$$

This is just the encoding of booleans in §2.3.4. If the target calculus is the untyped asynchronous π-calculus, this encoding is sound but not fully abstract. As in the case of encoding synchrony into asynchrony, no fully abstract encoding of binary branching has been published at the time of writing. We conjecture that the linear types of [124] will again easily lead to full abstraction.

Inspired by linear logic [41], Honda [55] and Vasconcelos [114] were the first to propose and develop external choice as a primitive for name passing interaction. Lopez [69] considers implementation issues arising from external choice.

2.5 Matching and Mismatching

Two other extension that are often considered are matching and mismatching. Matching allows to compare two names for equality while mismatching is the nega-

tion of matching and allows to test names for being distinct. Both can be useful, in particular for axiomatisations of equivalences. Syntactically, the *asynchronous π-calculus with matching* is given by the following grammar.

P	::=	$\overline{x}\langle\vec{y}\rangle$	output
	|	$x(\vec{y}).\mathsf{P}$	input prefix
	|	$!x(\vec{y}).\mathsf{P}$	lazy replication
	|	$[x=y]\mathsf{P}$	matching
	|	$\mathsf{P} \mid \mathsf{Q}$	parallel composition
	|	$(\nu x)\mathsf{P}$	hiding
	|	$\mathsf{0}$	inaction

The *asynchronous π-calculus with mismatching* is obtained by the previous grammar where $[x=y]\mathsf{P}$ is replaced by $[x \neq y]\mathsf{P}$. Of course it is also possible to consider calculi that combine matching and mismatching.

The reduction semantics of matching is obtained by just adding

$$[x=x]\mathsf{P} \equiv \mathsf{P}$$

to the axioms of the structural congruence. For the labelled transitions we need to add

$$(\textsc{Match}) \quad \frac{\mathsf{P} \xrightarrow{l} \mathsf{Q}}{[x=x]\mathsf{P} \xrightarrow{l} \mathsf{Q}}$$

The modifications for asynchrony and mismatching are similar.

Both operations, like output prefixing and sums are powerful in that they distinguish various equivalences that coincide in the asynchronous π-calculus.

2.6 HOπ

We have emphasised that π-calculi communicate entities without (significant) internal structure. This simplifies mathematical development and implementations but it is in stark contrast with the program passing approach to computation that, in the guise of functional programming, has dominated mathematical accounts of computation for two decades. As the name suggests, program passing is characterised by computational agents that exchange programs as their main computational mechanism. λ-calculi are the most prominent program passing formalisms, but concurrency theory, too, has some on offer, Boudol's γ-calculus [23] for example or CHOCS [112]. Both implement interaction of parallel agents as process passing.

Naturally the question arises as to the relative expressiveness of process passing and name passing. Clearly the latter is a special case of the former because names can be seen as certain trivial programs that cannot do anything apart from communicating their identity. The converse is more involved: can process passing

be reduced to name passing? In 1992 Sangiorgi answered the question to the affirmative. He showed that a π-calculus with process passing can be fully-abstractly encoded into a π-calculus without by replacing process passing with passing pointers (names) to processes [100, 103]. Surprisingly, this encoding is fully abstract without the target calculus being typed. This suggests that – unlike many other embeddings into π-calculi– the granularity of the computational mechanism is not changed by the encoding.

Nevertheless, process passing is often a convenient abstraction and we shall use it later in this text, albeit in a very constrained from.

There is considerable leeway in how to add process passing to π-calculi, but, as it does not play a mayor role in our subsequent exposition, we shall present it in what might be its simplest form. Processes are give by the following grammar.

P	::=	$x(\vec{y}).\mathsf{P}$	input prefix
	|	$!x(\vec{y}).\mathsf{P}$	lazy replication
	|	$\overline{x}\langle\vec{y}\rangle$	output
	|	$\overline{x}\langle\vec{\mathsf{P}}\rangle$	output of processes
	|	x	starting a received process
	|	$\mathsf{P} \mid \mathsf{Q}$	parallel composition
	|	$(\nu x)\mathsf{P}$	hiding
	|	$\mathsf{0}$	inaction

With $\overline{x}\langle\vec{\mathsf{P}}\rangle$ we add the ability to output a vector of processes. The stand-alone name x launches a received process. It might be confusing to use bound names to be instantiated with names and processes and we could choose to have different syntactic categories of bound names instead, to distinguish the two, but for easy of exposition we have chosen not to and rely on sorting instead for disambiguation: for example

$$\overline{x}\langle y\rangle \mid x(v).(v \mid \mathsf{P})$$

is ill sorted, while

$$\overline{x}\langle\mathsf{Q}\rangle \mid x(v).(v \mid \mathsf{P})$$

is not. The reduction semantics need only two additional rules.

$$\overline{x}\langle\vec{\mathsf{P}}\rangle \mid x(\vec{v}).\mathsf{Q} \to \mathsf{Q}\{\vec{\mathsf{P}}/\vec{v}\} \qquad \overline{x}\langle\vec{\mathsf{P}}\rangle \mid !x(\vec{v}).\mathsf{Q} \to \mathsf{Q}\{\vec{\mathsf{P}}/\vec{v}\} \mid !x(\vec{v}).\mathsf{Q}$$

The axioms for the structural congruence remain unchanged. Of course substitution in the above rule needs to avoid capture of bound names, but the variable convention applies just as it does in the first-order case. To compute the free and bound names

of higher-order processes we need additional clauses.

$$\begin{aligned} \mathsf{fn}(x) &= \{x\} \\ \mathsf{bn}(x) &= \emptyset \\ \mathsf{fn}(\overline{x}\langle\vec{\mathsf{P}}\rangle) &= \{x\} \cup \bigcup \mathsf{fn}(\mathsf{P}_i) \\ \mathsf{bn}(\overline{x}\langle\vec{\mathsf{P}}\rangle) &= \mathsf{bn}(\mathsf{P}_i) \end{aligned}$$

We omit the labelled transitions as they are quite involved. Please consult [100, 103] for details, including full-abstraction proofs.

2.7 Variants (2): Restrictions

All variants of π-calculi discussed in the previous section have been *extensions*. It can also be fruitful to go the other way and restrict π-calculi. This cannot mean to remove further operators from, for example, the asynchronous π-calculus, for that would surely damage its expressivity: removal of any operator apart from restriction would probably render the resulting calculus less than Turing-Universal, while getting rid of restriction would destroy compositionality. What is often the case, though, is that we don't need the whole set of processes generated by the grammar of the chosen calculus. Restrictions *systematically* identify a proper subset of processes without compromising expressivity. This is usually done by typing. The main advantage of such restrictions is that the subcalculi have nicer theoretical properties, i.e., more equations hold because fewer observers exist that could distinguish processes. We shall now present the two most well known examples of such restrictions, locality and internal mobility. They will also play a role later.

In both cases the reduction and transition semantics carry over unchanged, but work on a smaller set of processes. We will not present typing systems that would ensure the respective properties because setting up the formalisms to do that would be too much work for a text not otherwise concerned with types.

2.7.1 Locality

To understand locality, think about your machine. It is probably connected to some Ethernet based LAN, which is in turn connected to the Internet. This means that your computer will have one or more IP addresses to identify its points of interaction with other computers. What is peculiar about IP addresses is that they are *fixed*. They cannot be changed (easily). In particular, you cannot make one of your IP addresses dependent on data that has just come in from the Internet.

Of course things are more complicated than this. One can always change IP addresses of a machine, but that would in itself not be very useful, unless the entire Internet was notified about the address change. Otherwise routing tables would

not contain up-to-date information that would allow to route messages properly. This is a time consuming process, taking several hours if not longer. As a matter of fact, things are even more complicated because some address changes can be carried out locally, for example from 236.72.1.61 to 236.72.1.62, provided both addresses already live in the same broadcast LAN and 236.72.1.62 is not already used. Since all messages going to 236.72.1.62 will be broadcast, either by the gateway router or by some other machine on the LAN, delivery to the new address is no problem. But what certainly does not work is to assign an *arbitrary* IP address to a computer and expect that instantly messages from *anywhere* on the net are delivered. It is possibly rather crude, but nevertheless illuminating to model this behaviour as π-calculus-processes, taking names to stand for IP addresses: this boils down to prohibiting input subject to be input bound. This means not allowing processes like $x(v).v(w)$, because the input subject v is bound by $x(v)$. π-calculi where input subjects are never bound by inputs are called *local*. It is easy to define typing systems such that typability guarantees locality. Merro and Sangiorgi [72] investigates locality and concludes that locality does not significantly restrict the expressive power of π-calculi. Locality will be crucial in later chapters.

2.7.2 Internal Mobility

Much of the expressivity π-calculi gain over their predecessors derives from the ability to dynamically create private links between processes by way of scope extending outputs. What happens if the only kind of interaction permitted is the creation of private links? What happens, in other words, if we only have bound output? *Bound outputs* are outputs where every carried name induces a scope extension, as it does in

$$(\nu yz)(\overline{x}\langle yz\rangle \mid \mathsf{P}) \mid x(vw).\mathsf{Q} \rightarrow (\nu yz)(\mathsf{P} \mid \mathsf{Q}\{yz/vw\}).$$

The next example shows an output that is not bound.

$$(\nu y)(\overline{x}\langle yz\rangle \mid \mathsf{P}) \mid x(vw).\mathsf{Q} \rightarrow (\nu y)(\mathsf{P} \mid \mathsf{Q}\{yz/vw\})$$

This is not considered a bound output, despite extending the scope of y, because the environment may already have z as an interaction point and use it to interfere with P or Q or their descendants. π-calculi where all outputs are bound are said to exhibit *internal mobility* and have been studied in some detail by Sangiorgi [101, 102]. He shows that almost all of π-calculi expressive power is already contained in their internally mobile cores, while some of the complications of reasoning about π-calculi stem from the ability to do free or non-bound outputs.

While enjoying simpler theoretical properties, programming restricted to bound outputs is cumbersome because *passing on information* is no longer straightforward: consider a typical server (for example computing a function)

$$!x(\vec{y}z).\mathsf{P},$$

invoked by clients which supply access points $\vec{y}$ to the arguments of the service in addition to a return channel z for the delivery of the eventual result. As program synthesis is often compositional, all P might do is pass on the $\vec{y}$s and z to some appropriate sub-servers. Alas, this cannot be expressed directly using just bound outputs because in P, y and z would be free. We would have to use forwarders or copycat processes [5, 15, 16, 124] instead which relay information between interaction points. Such processes are quite simple but their proliferation can make processes hard to read. Free outputs seem to be a more succinct representation of information transmission. Consequently, most of our examples will use free outputs to aid readability. All free outputs we use can be replaced by bound ones with appropriate forwarding.

2.8 Concluding Remark

Much more could be said about π-calculi, but we do not aim to be comprehensive, so we stop right here.

Chapter 3

The Two Phase Commit Protocol

This chapter introduces the Two Phase Commit Protocol, an important distributed algorithm, and sketches its correctness proof.

3.1 Introduction

The *Two Phase Commit Protocol* [19, 47] (from now on 2PCP for short) is an important and ubiquitous distributed algorithm. Its role in this thesis is to act as a test of the π-calculus extensions to be introduced. If they are to be of any value, they better be able to express the 2PCP. Of course, the real test is not just its expression, but also whether it can be implemented in a way that facilitates feasible correctness proofs. Although the 2PCP is trivial, at least in its algorithmic core, as we shall see soon, its distribution makes verification a challenge. So much so in fact, that [13, 14] and their elaboration in the present text seem to have been the first attempts at doing so in a rigorously formal way.

What is the 2PCP? A short but pretty accurate answer is that it is a distributed computation of the n-ary logical `and`: each of the n processes involved votes for either `true` or `false`. The result of this voting process is that all process agree on `true` exactly when all processes have voted `true`. Otherwise they agree on `false`.

This description is not only fairly accurate but also problematic because of distribution. What does it mean to vote or to agree? In a distributed setting, messages carrying votes can be lost or duplicated. They might arrive late. What happens if a process decides to vote a certain way, but crashes before being able to send a vote to another process?

As we will see, the possibility of failures makes the 2PCP somewhat more complicated than one would expect from its description as implementing logical conjunction. Its correctness proof is a bit of a nightmare, due to the veritable error-masking

scaffold that ensures consistent agreement despite the possible failure scenarios. To help the reader understand the 2PCP, this chapter introduces the algorithm in a drastically simplified form: no failures at all. We sketch the correctness proof for this simple version, because its macro-structure will be unaffected by embellishing the core protocol for survival in the distributed world. What changes in the proof is that the possibility of failures makes the individual steps more complicated.

3.2 The Algorithmic Core of the 2PCP

For our purposes, it might be convenient to think of the 2PCP as being made up of $n+1$ processes: the *coordinator* C and n *participants* P_i who communicate with each other on the private channels $vote_i$ and dec_i.

$$\mathsf{2PCP} = (\nu \vec{vote})(\nu \vec{dec})(\mathsf{C} \mid \mathsf{P}_1 \mid ... \mid \mathsf{P}_n)$$

The participants supply the votes for `true` or `false` to the coordinator, with each process having exactly one vote. The coordinator receives all the votes, adds its own, decides on the overall vote and notifies all participants of the outcome. Each participant P_i then relays that choice to the outside world by repeatedly sending either on the channel $abort_i$ or on $commit_i$ with the former corresponding to `false` and the latter to `true`. This choice of terminology (*abort*, *commit*) betrays the 2PCP's origins in transaction processing, where participants are transactions, the coordinator is a database management system and the purpose of the protocol is to aid enforcing atomicity of the database by allowing transactions to commit only if all participating transactions do the same. Of course in a real database, the participants will execute more complex computations upon commitment or abort, but this is not what we seek to model here and hence omitted.

In the idealised form we present it here, the 2PCP does not use scope extension and is finite state. Real implementations would usually lift both restrictions: the 2PCP would work for an arbitrary number of participants, not just for some fixed n. It would feature an initial joining stage where interested processes could register their participation in the protocol. This would mean we had infinitely many states. The joining process would also often dynamically create private channels between the coordinator and participants for communication of votes and the like. In a π-calculus setting that could be conveniently modelled by bound outputs.

Participants

Participants are straightforward. Their first task is to decide nondeterministically on their vote.

$$\mathsf{P}_i = \mathsf{P}_i^{abort} \oplus \mathsf{P}_i^{commit}$$

This vote is then communicated to the coordinator on the channel $vote_i$ using branching output. Branching simplifies presentation and reasoning, but could be removed, as described in §2.4.4. If P_i decides to vote `false`, here implemented by branching "to the right", it has no need to wait for the coordinator's decision, for that can only be `false`. Instead the process aborts straightaway.

$$\mathsf{P}_i^{abort} = \overline{vote_i}\mathsf{right} \,|\, !\overline{abort_i}$$

If a participant decides to vote `true`, represented by a branch to the left, it has to wait for the coordinator's decision, as other votes may cause an overall abort.

$$\begin{aligned} \mathsf{P}_i^{commit} &= \overline{vote_i}\mathsf{left} \,|\, \mathsf{P}_i^{wait} \\ \mathsf{P}_i^{wait} &= dec_i[!\overline{commit_i}, !\overline{abort_i}] \end{aligned}$$

The coordinator's decision is received by the participant on dec_i, using branching.

The Coordinator

In the absence of failures, the coordinator is only a tiny bit more complicated. It also starts by deciding on its vote.

$$\mathsf{C} = \mathsf{C}^{abort} \oplus \mathsf{C}_{pre}^{commit}$$

If that decision is towards aborting, the coordinator ignores the participants' votes and instructs them to abort straightaway.

$$\begin{aligned} \mathsf{C}^{abort} &= \Pi_{i=1}^{n} \mathsf{C}_i^{abort} \\ \mathsf{C}_i^{abort} &= \overline{dec_i}\mathsf{right} \end{aligned}$$

Otherwise, it collects all the votes and computes their logical conjunction.

$$\begin{aligned} \mathsf{C}_{pre}^{commit} &= (\nu \vec{c}a)(\mathsf{C}^{wait} | \, \mathsf{C}^{and} \mid \mathsf{C}^{or}) \\ \mathsf{C}^{wait} &= \Pi_{i=1}^{n} \mathsf{C}_i^{wait} \\ \mathsf{C}_i^{wait} &= vote_i[\overline{c_i}, \overline{a}] \\ \mathsf{C}^{and} &= c_1.c_2...c_n.\mathsf{C}_{final}^{commit} \\ \mathsf{C}_{final}^{commit} &= \Pi_{i=1}^{n} \mathsf{C}_i^{commit} \\ \mathsf{C}_i^{commit} &= \overline{dec_i}\mathsf{left} \\ \mathsf{C}^{or} &= a.\mathsf{C}^{abort} \end{aligned}$$

Remarks on the Underlying Calculus

The language we have used to express the 2PCP is not the asynchronous π-calculus of §2.2.1, but a variant with binary branching, cf. §2.4.4 and a form of lazy replication that allows not only input-guarded replication but also output replication, as described in §2.4.1. Although this calculus is slightly unconventional, it is also highly convenient for the task at hand. Moreover, all relevant results discussed in Chapter 2 also hold for this calculus.

It would not be a problem to remove branching from this description of the 2PCP, at the price of making the implementation harder to understand and its verification more cumbersome. As branching does not cause significant problems with the kind of theory we are going to develop in subsequent chapters, is appears that its advantages outweigh the disadvantages of additional syntax and inference rules. The same is true of output guarded replication and recursive equations.

3.3 Correctness of the 2PCP

Before we can prove the 2PCP correct, we have to decide on what its correctness means formally. Fortunately, this is *almost* easy. In essence, we want to avoid some participants committing while others don't. The next theorem expresses this succinctly, but first a definition.

$$\begin{aligned} \mathsf{Commit} &= \Pi_{i=1}^{n} !\overline{commit_i} \\ \mathsf{Abort} &= \Pi_{i=1}^{n} !\overline{abort_i} \end{aligned}$$

Now a key correctness criterion can be expressed succinctly.

THEOREM 12 $\mathsf{2PCP} \approx_a \mathsf{Abort} \oplus \mathsf{Commit}$

Of course this theorem does not contain all one could reasonably require of a suitable correctness criterion for the 2PCP, because it does not mandate the protocol to compute the logical conjunction of all the participants' votes. An implementation that would force all participants to agree on the P_2's decision would also meet Theorem 12. But it is easy to fix this shortcoming.

THEOREM 13

1. $(\nu \vec{vote})(\nu \vec{dec})(\mathsf{C}_{pre}^{commit} \mid \mathsf{P}_1^{commit} \mid ... \mid \mathsf{P}_n^{commit}) \approx_a \mathsf{Commit}$.
2. *If* $\mathsf{P}'_i \in \{\mathsf{P}_i^{abort}, \mathsf{P}_i^{commit}\}$, *then* $(\nu \vec{vote})(\nu \vec{dec})(\mathsf{C}^{abort} \mid \mathsf{P}'_1 \mid ... \mid \mathsf{P}'_n) \approx_a \mathsf{Abort}$.
3. *If* $\mathsf{P}'_i \in \{\mathsf{P}_i^{a}, \mathsf{P}_i^{commit}\}$ *and for at least one* $i_0 \in \{1, ..., n\}$ $\mathsf{P}'_{i_0} = \mathsf{P}_{i_0}^{abort}$, *then*

$$(\nu \vec{vote})(\nu \vec{dec})(\mathsf{C}_{pre}^{commit} \mid \mathsf{P}'_1 \mid ... \mid \mathsf{P}'_n) \approx_a \mathsf{Abort}.$$

Unfortunately, this is still not enough. To nail down the correctness of the 2PCP, we'd also have to ensure that nothing goes wrong in the transition from 2PCP to the processes on the left in the equations of Theorem 13. After all, there is no apriori reason that all transitions from 2PCP must evolve only via those processes. It is possible and mathematically trivial to remedy this shortcoming, but formally rather inconvenient. Since the definition of 2PCP is transparent enough to allow anyone even mildly conversant with π-calculi to instantly see that our implementation does not suffer from this shortcoming, we chose to stick with just establishing the simpler and substantive, yet merely approximative correctness, expressed by Theorems 12 and 13.

The rest of this chapter sketches a proof of both theorems. Most of the reasoning is standard π-calculus technology, so many details are omitted. The proof itself involves a classic case of Divide-and-Conquer. First we show that in certain circumstances

$$\left.\begin{array}{l}\mathsf{P} \mid \mathsf{Q} \approx_a \mathsf{S} \\ \mathsf{P} \mid \mathsf{R} \approx_a \mathsf{S} \oplus \mathsf{T}\end{array}\right\} \Rightarrow \mathsf{P} \mid (\mathsf{Q} \oplus \mathsf{R}) \approx_a \mathsf{S} \oplus \mathsf{T}$$

and then use this fact to show by partial induction that (roughly)

$$C \mid \mathsf{P}_1 \mid ... \mid \mathsf{P}_i \mid \mathsf{P}_{i+1} \approx_a \mathsf{Abort}_{i+1} \oplus \mathsf{Commit}_{i+1}$$

can be inferred from

$$C \mid \mathsf{P}_1 \mid ... \mid \mathsf{P}_i \mid \mathsf{P}^c_{i+1} \approx_a \mathsf{Abort}_{i+1} \oplus \mathsf{Commit}_{i+1} \text{and} C \mid \mathsf{P}_1 \mid ... \mid \mathsf{P}_i \mid \mathsf{P}^a_{i+1} \approx_a \mathsf{Abort}_{i+1}.$$

Here $(\mathsf{Abort}_i)_{i=1}^{n+1}$ and $(\mathsf{Commit}_i)_{i=1}^{n+1}$ are suitable sequences of processes such that $\mathsf{Abort}_{n+1} = \mathsf{Abort}$ and $\mathsf{Commit}_{n+1} = \mathsf{Commit}$. The limit case $i = n$ is Theorem 12 and some of the intermediate steps in its proof specialise to Theorem 13.

Some Useful Definitions and Lemmas

If key steps of the proof infer properties of the 2PCP with $i+1$ participants from the 2PCP with i participants, doesn't the protocol's outermost restrictions, hiding the channels for voting and communication of the decision, stand in the way of a straightforward induction? After all, we induce on i with the processes obtained by taking off these restrictions. Yes, but the next few definitions provide tools for ignoring outermost restrictions.

DEFINITION 11 Let S be a set of names. We call actions l with $\mathsf{fn}(l) \cap S \neq \emptyset$ *S-hidden*. A process is *static* if all of its or its descendants outputs are free outputs.

A binary relation $\mathcal{R}$ on processes is an *asynchronous S-bisimulation* if $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ implies that P as well as Q are static and

- whenever $\mathsf{P} \xrightarrow{l}_a \mathsf{P}'$ and $\mathsf{fn}(l) \cap S = \emptyset$, then $\mathsf{Q} \overset{\hat{l}}{\twoheadrightarrow}_a \mathsf{Q}'$ for some Q' such that $(\mathsf{P}', \mathsf{Q}') \in \mathcal{R}$,

- and vice versa.

By $\approx_a^S$ we denote the largest S-bisimulation. The definitions of *strong asynchronous S-bisimulation*, *S-bisimulation*, *strong S-bisimulation*, $\sim_a^S$, $\approx^S$ and $\sim^S$ are similar.

The next two theorems summarise some useful properties of the equivalences just defined and how they relate to the standard equivalences.

THEOREM 14 *$\approx^S \subseteq \approx_a^S$, $\sim^S \subseteq \approx^S$ and $\sim_a^S \subseteq \approx_a^S$.*

PROOF: Straightforward. □

THEOREM 15 *Let $\mathcal{R}^S$ be one of $\approx^S, \approx_a^S, \sim^S, \sim_a^S$.*

1. *$\mathsf{P}\,\mathcal{R}^S\,\mathsf{Q}$ implies $(\nu x)\mathsf{P}\,\mathcal{R}^{S\setminus\{x\}}\,(\nu x)\mathsf{Q}$. In addition: $\mathcal{R}^\emptyset \subseteq \mathcal{R}$.*
2. *If $S \subseteq T$, then $\mathcal{R}^S \subseteq \mathcal{R}^T$.*
3. *If $\mathsf{P}\,\mathcal{R}^S\,\mathsf{Q}$ and $\mathsf{fn}(\mathsf{R}) \cap S = \emptyset$ then $\mathsf{P} \mid \mathsf{R}\,\mathcal{R}^S\,\mathsf{Q} \mid \mathsf{R}$.*

PROOF: Straightforward. □

The next theorem formalises the Divide-and-Conquer principle alluded to above. For simplicity and to facilitate comparison with the use of Divide-And-Conquer in the versions of the 2PCP correctness proof to come later, we present it in 3 different forms, although unification isn't difficult.

THEOREM 16 *Let S a set of names.*

1. *Let $(\mathsf{P}_i)_{i\in I}$, $(\mathsf{Q}_j)_{j\in J}$, $(\mathsf{R}_j)_{j\in J}$ and $(\mathsf{S}_i)_{i\in I}$ be collections of processes such that for all $i \in I$ and $j \in J$:*
   - $\mathsf{P}_i \mid \mathsf{Q}_j \approx^S \mathsf{S}_i$,
   - $\mathsf{P}_i \mid \mathsf{R}_j \approx^S \mathsf{S}_i$,
   - *If $\mathsf{P}_i \xrightarrow{l}_a \mathsf{P}$ is not S-hidden, then $\mathsf{P} \equiv \mathsf{P}_k$ for some $k \in I$ and $\mathsf{S}_i \xrightarrow{l}_a \mathsf{S}_k$.*

   Then $\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_j) \approx^S \mathsf{S}_i$.
2. *Let $\{\mathsf{P}_i\}_{i\in I}$, $\{\mathsf{Q}_j\}_{j\in J}$, $\{\mathsf{R}_k\}_{k\in K}$, $\{\mathsf{S}_s\}_{s\in S}$, $\{\mathsf{T}_t\}_{t\in T}$ be processes. Assume $f : I\times J \to S$ and $g : I\times K \to T$ are functions such that for all $i \in I, j \in J, k \in K$:*
   - $\mathsf{P}_i \mid \mathsf{Q}_j \approx^S \mathsf{S}_{f(i,j)}$,
   - $\mathsf{P}_i \mid \mathsf{R}_k \approx^S \mathsf{T}_{g(i,k)}$
   - *whenever $\mathsf{P}_i \xrightarrow{l}_a \mathsf{P}$ for some action l that is not S-hidden, then one of the following is true.*

– $\mathsf{S}_{f(i,j)} \oplus \mathsf{T}_{g(i,k)} \stackrel{\hat{l}}{\twoheadrightarrow}_a \mathsf{V}$ *for some process* V *with* $\mathsf{P} \mid (\mathsf{Q}_j \oplus \mathsf{R}_k) \approx^S \mathsf{V}$.

– $\mathsf{P} \equiv \mathsf{P}_{i_0}$ *for some* $i_0 \in I$ *such that* $\mathsf{S}_{f(i,j)} \oplus \mathsf{T}_{g(i,k)} \stackrel{\hat{l}}{\twoheadrightarrow}_a \mathsf{R}_{f(i_0,j)}$.

Then $\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_k) \approx^S \mathsf{S}_{f(i,j)} \oplus \mathsf{T}_{g(i,k)}$ *for all* i, j, k.

3. *Assume that for all* $i \in I$, $\mathsf{P}_i, \mathsf{Q}_i, \mathsf{R}_i, \mathsf{S}_i$ *and* T_i *are processes such that*

- *if* x *is a free name in any of the processes mentioned about, but* $x \notin S$, *then* x *is not the subject of an input.*
- $\mathsf{P}_i \mid \mathsf{R}_i \approx_a^S \mathsf{S}_i \oplus \mathsf{T}_i$,
- $\mathsf{Q}_i \mid \mathsf{R}_i \approx_a^S \mathsf{T}_i$ *and*
- *whenever* l *is not* S*-hidden and* $\mathsf{R}_i \stackrel{l}{\longrightarrow}_a \mathsf{R}$, *then a process* U *exists such that* $\mathsf{S}_i \oplus \mathsf{T}_i \stackrel{l}{\twoheadrightarrow}_a \mathsf{U}$, $\mathsf{P}_i \mid \mathsf{U} \approx_a^S \mathsf{S}_i$ *and* $\mathsf{Q}_i \mid \mathsf{U} \approx_a^S \mathsf{T}_i$.

Then $(\mathsf{P}_i \oplus \mathsf{Q}_i) \mid \mathsf{R}_i \approx_a^S \mathsf{S}_i \oplus \mathsf{T}_i$.

PROOF: We begin with (2). The proof of (1) is similar but easier. We define $\mathcal{R}$ by

$$\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_k) \;\; \mathcal{R} \;\; \mathsf{S}_{f(i,j)} \oplus \mathsf{T}_{g(i,k)}$$

for all i, j, k. Then we have the following relevant transitions.

- $\mathsf{S}_{f(i,j)} \oplus \mathsf{T}_{g(i,k)} \stackrel{\tau}{\rightarrow} \mathsf{S}_{f(i,j)}$. It is matched by $\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_k) \stackrel{\tau}{\rightarrow} \mathsf{P}_i \mid \mathsf{Q}_j$. The other related choice is matched similarly.
- $\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_k) \stackrel{\tau}{\rightarrow} \mathsf{P}_i \mid \mathsf{Q}_j$ is matched by $\mathsf{S}_{f(i,j)} \oplus \mathsf{T}_{g(i,k)} \stackrel{\tau}{\rightarrow} \mathsf{S}_{f(i,j)}$. The choice towards R_k is again similar.
- If $\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_k) \stackrel{l}{\rightarrow} \mathsf{P} \mid (\mathsf{Q}_j \oplus \mathsf{R}_k)$ because $\mathsf{P}_i \stackrel{l}{\rightarrow} \mathsf{P}$, then we have two cases. If $\mathsf{S}_{f(i,j)} \oplus \mathsf{T}_{g(i,k)} \stackrel{\hat{l}}{\twoheadrightarrow}_a \mathsf{V}$ for some process V with $\mathsf{P} \mid (\mathsf{Q}_j \oplus \mathsf{R}_k) \approx^S \mathsf{V}$, then we already have the matching transition. Otherwise, by assumptions, $\mathsf{P} \equiv \mathsf{P}_{i_0}$ for some $i_0 \in I$ such that $\mathsf{S}_{f(i,j)} \oplus \mathsf{T}_{g(i,k)} \stackrel{\hat{l}}{\twoheadrightarrow}_a \mathsf{R}_{f(i_0,j)}$.

Hence $\mathcal{R} \cup \approx^S$ is a S-bisimulation.

The proof of 3 is similar to the previous one, except that we have to establish asynchronous, not synchronous S-bisimilarity. That means, we have to saturate the candidate relation with appropriate outputs. Because of the restrictions on free names in the processes under consideration, this does not produce interesting or problematic new transitions. □

LEMMA 3 1. *Assume that* $x \in S \setminus \mathsf{fn}(\mathsf{P})$.

- $\mathsf{P} \mid x[(\vec{v}).\mathsf{Q} \;\&\; (\vec{w}).\mathsf{R}] \mid \overline{x}\mathsf{left}\langle\vec{y}\rangle \approx^S \mathsf{P} \mid \mathsf{Q}\{\vec{y}/\vec{v}\}$.

- $\mathsf{P} \mid x[(\vec{v}).\mathsf{Q} \;\&\; (\vec{w}).\mathsf{R}] \mid \overline{x}\mathsf{right}\langle\vec{y}\rangle \approx^S \mathsf{P} \mid \mathsf{R}\{\vec{y}/\vec{w}\}$.
- $\mathsf{P} \mid x(\vec{v}).\mathsf{Q} \mid \overline{x}\langle\vec{y}\rangle \approx^S \mathsf{P} \mid \mathsf{Q}\{\vec{y}/\vec{v}\}$.

2. *Assume that* $x \in S \setminus \mathsf{fn}(\mathsf{P})$.

   - $\mathsf{P} \mid \overline{x}\mathsf{left}\langle\vec{y}\rangle \sim^S \mathsf{P}$.
   - $\mathsf{P} \mid \overline{x}\mathsf{right}\langle\vec{y}\rangle \sim^S \mathsf{P}$.
   - $\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \sim^S \mathsf{P}$.

3. *Let* a, $\vec{x} = \langle x_0, ..., x_{n-1}\rangle$ *and* $\vec{y} = \langle y_0, ..., y_{n-1}\rangle$ *be fresh. Let* $0 \leq i_0 < n$. *If* $y_0, y_1, ..., y_{i_0} \in S$, *then*

$$(\nu\vec{x}a)(x_0...x_{n-1}.\mathsf{P} \mid \Pi_{i=0, i\neq i_0}^{n} y_i[\overline{x_i}, \overline{a}] \mid \mathsf{Q}) \approx_a^S \mathsf{Q}.$$

Proof: Straightforward. □

The Core of the Proof

Before delving into the details, let's illustrate the proof from another angle. Using an information flow metaphor, the failure-free 2PCP has 3 essential ingredients.

- The $n + 1$ internal sums that are the protocol's only source of uncertainty or information. This information flows independently from each participant to the coordinator, via dedicated channels.
- In the coordinator, this information becomes entangled and compressed into one single bit, the overall decision.
- This decision flows back from the coordinator to all participants, again on dedicated channels.

The proof can be understood as following this information flow: the non-determinism is pushed, using the Divide-and-Conquer Theorem 16, to the coordinator and then back to the participants. The entanglement process in the coordinator is reflected in the proof by compressing the $n + 1$ internal choices into just one.

The present author passionately believes that this correspondence between information flow in a process and the structure proofs about its properties is an instance of a general phenomenon that relates information flow and proofs. Alas, distilling this intuition into something mathematically communicable has not been successful so far.

We start with defining abbreviations for some of the processes that are vital to the correctness proof, but have not been named already.

DEFINITION 12 Let $k \in \{1,...,n\}$ and $S,T \subseteq \{1,...,n\}$.

$$\begin{array}{rcl}
\mathsf{C}^{commit}_{pre,\,\mathrm{k}} & = & (\nu \vec{c}a)(\Pi^n_{j=k}\mathsf{C}^{wait}_j \mid \mathsf{C}^{and}_k \mid \mathsf{C}^{or}) \\
\mathsf{C}^{and}_k & = & c_k.....c_n.\mathsf{C}^{commit}_{final} \\
\mathsf{C}^{and}_{n+1} & = & \mathsf{C}^{commit}_{final} \\
\mathsf{D}^c_{S,T,k} & = & (\nu \vec{c}a)(\Pi_{j\in S}\mathsf{C}^{wait}_j \mid \Pi_{j\in T}\overline{c_j} \mid \mathsf{C}^{and}_k \mid \mathsf{C}^{or}) \mid \Pi_{i\in T\cup\{1,...,k-1\}} dec_i[!\overline{commit_i}, !\overline{abort_i}] \\
\mathsf{D}^c_{S,T,n+1} & = & \Pi^n_{i=1}!\overline{commit_i} \\
\mathsf{D}^a_{S,T} & = & \Pi_{i\in S}\overline{dec_i}\mathsf{right} \mid \Pi_{i\in T}!\overline{abort_i}
\end{array}$$

The process $\mathsf{D}^c_{S,T,k}$ consists of a bunch of participants (their indices form the set $T \cup \{1,\ldots,k-1\}$) waiting for the coordinator's decision (hence they must all have voted for commitment) together with the corresponding residual of the coordinator. The set S contains all the indices of processes whose votes have not yet been received. The process $\mathsf{D}^a_{S,T}$ can be though of as the remaining decision messages from the aborting coordinator (indices given by S) together with aborting participants (indices in T).

To be able to reason without being unduly hindered by the outermost restrictions of the 2PCP, we use S-bisimulations. The next definition fixes the sets of names that we will use for this purpose.

DEFINITION 13 Let $S \subseteq \{1,...,n\}$.

$$H_S = \bigcup_{i\in S}\{vote_i, dec_i\} \qquad\qquad H_i = H_{\{1,...,i\}}$$

We can now state and verify the first key result about the 2PCP. It essentially says that if any decision is made towards voting to abort, then all processes will abort.

LEMMA 4 *Let $i \in \{1,...,n\}$ and $S,T \subseteq \{1,...,n\}$ such that $i \in S$ and $S \cap T = \emptyset$.*

1. $\mathsf{D}^a_{S,T} \mid \mathsf{P}^{abort}_i \approx^{H_{\{i\}}} \mathsf{D}^a_{S\setminus\{i\},T\cup\{i\}}$.
2. $\mathsf{D}^a_{S,T} \mid \mathsf{P}^{commit}_i \approx^{H_{\{i\}}} \mathsf{D}^a_{S\setminus\{i\},T\cup\{i\}}$.
3. $\mathsf{D}^a_{S,T} \mid \mathsf{P}_i \approx^{H_{\{i\}}} \mathsf{D}^a_{S\setminus\{i\},T\cup\{i\}}$.

PROOF: We begin with (1).

$$\begin{array}{rcl}
\mathsf{D}^a_{S,T} \mid \mathsf{P}^{abort}_i & \equiv & \Pi_{j\in S}\overline{dec_j}\mathsf{right} \mid \Pi_{j\in T}!\overline{abort_j} \mid \overline{vote_i}\mathsf{right} \mid !\overline{abort_i} \\
& \equiv & \Pi_{j\in S}\overline{dec_j}\mathsf{right} \mid \Pi_{j\in T\cup\{i\}}!\overline{abort_j} \mid \overline{vote_i}\mathsf{right} \\
& \sim^{H_{\{i\}}} & \Pi_{j\in S}\overline{dec_j}\mathsf{right} \mid \Pi_{j\in T\cup\{i\}}!\overline{abort_j} \\
& \sim^{H_{\{i\}}} & \Pi_{j\in S\setminus\{i\}}\overline{dec_j}\mathsf{right} \mid \Pi_{j\in T\cup\{i\}}!\overline{abort_j} \\
& \equiv & \mathsf{D}^a_{S\setminus\{i\},T\cup\{i\}}.
\end{array}$$

Here we are using Lemma 3.2. For (2) we proceed in a similar fashion.

$$\begin{array}{rcl}
\mathsf{D}^a_{S,T} \mid \mathsf{P}_i^{commit} & \equiv & \Pi_{j\in S}\overline{dec_j}\mathsf{right} \mid \Pi_{j\in T}!\overline{abort_j} \mid \overline{vote_i}\mathsf{left} \mid dec_i[!\overline{commit_i}, !\overline{abort_i}] \\
& \sim^{H_{\{i\}}} & \Pi_{j\in S}\overline{dec_j}\mathsf{right} \mid \Pi_{j\in T}!\overline{abort_j} \mid dec_i[!\overline{commit_i}, !\overline{abort_i}] \\
& \approx^{H_{\{i\}}} & \Pi_{j\in S\setminus\{i\}}\overline{dec_j}\mathsf{right} \mid \Pi_{j\in T}!\overline{abort_j} \mid !\overline{abort_i} \\
& \equiv & \Pi_{j\in S\setminus\{i\}}\overline{dec_j}\mathsf{right} \mid \Pi_{j\in T\cup\{i\}}!\overline{abort_j} \\
& \equiv & \mathsf{D}^a_{S\setminus\{i\},T\cup i}
\end{array}$$

This derivation uses Lemma 3.1 and 3.2. It remains to verify (3). This is easy, because if $\mathsf{D}^a_{S,T} \xrightarrow{l} \mathsf{D}'$ where l is not $H_{\{i\}}$-hidden, then either

- $l = \overline{abort_j}$ for some $j \in T$, i.e. $j \neq i$, and $\mathsf{D}^a_{S,T} \equiv \mathsf{D}'$ and hence also $\mathsf{D}^a_{S\setminus\{i\},T\cup\{i\}} \xrightarrow{l} \mathsf{D}^a_{S\setminus\{i\},T\cup\{i\}}$; or
- $l = \overline{dec_j}\mathsf{right}$ for some $j \in S \setminus \{i\}$ and $\mathsf{D}' \equiv \mathsf{D}^a_{S\setminus\{j\},T}$. In this case

$$\mathsf{D}^a_{S\setminus\{i\},T\cup\{i\}} \xrightarrow{l} \mathsf{D}^a_{S\setminus\{i,j\},T\cup\{i\}}.$$

This means we can apply Theorem 16.1. □

We must establish a similar result for when one process casts its vote towards committing. This is more involved since such a decision removes the overall uncertainty only if all other processes have also already made the same decision. Otherwise the uncertainty must be preserved. The definition below proposes some technical tools that will help us with this.

Definition 14 Assume $k \in \{1, ..., n\}$ and $S \subseteq \{1, ..., n\}$ such that $j < k$ for all $j \in S$. Then we define

$$\mathsf{wait}_k(S) = \begin{cases} \emptyset & S = \emptyset \\ \mathsf{wait}_{k+1}(S \setminus \mathsf{min}(S)) & S \neq \emptyset, \mathsf{min}(S) = k \\ \{\mathsf{min}(S)\} \cup \mathsf{wait}_k(S \setminus \{\mathsf{min}(S)\}) & S \neq \emptyset, \mathsf{min}(S) > k \end{cases}$$

$$\mathsf{boundary}_k(S) = \begin{cases} k & S = \emptyset \\ \mathsf{boundary}_{k+1}(S \setminus \{\mathsf{min}(S)\}) & S \neq \emptyset, \mathsf{min}(S) = k \\ k & S \neq \emptyset, \mathsf{min}(S) > k. \end{cases}$$

Note that we can have $\mathsf{boundary}_k(S) = n + 1$.

We can now state the second key lemma in the proof. As with Lemma 4, it has three parts, with the last being derived from the first two by Divide-and-Conquer.

LEMMA 5 *Let $S,T \subseteq \{1,...,n\}$ and $i,k \in S$ such that $S \cap T = \emptyset$, $k \leq j$ for all $j \in S$ and for all $j \in T$: $k < j$. Abbreviate $H_{T\cup\{i,1,...,k-1\}}$ to $H_{T,i,k-1}$.*

1. $\mathsf{D}^c_{S,T,k} \,|\, \mathsf{P}_i^{abort} \approx^{H_{T,i,k-1}} \mathsf{D}^a_{\{k,...,n\}\setminus(T\cup\{i\}),T\cup\{1,...,k-1,i\}}$.
2. $\mathsf{D}^c_{S,T,k} \,|\, \mathsf{P}_i^{commit} \approx^{H_{T,i,k-1}} \mathsf{D}^c_{S\setminus\{i\},\mathsf{wait}_k(T\cup\{i\}),\mathsf{boundary}_k(T\cup\{i\})}$. *Here*
3. $\mathsf{D}^c_{S,T,k} \,|\, \mathsf{P}_i \approx^{H_{T,i,k-1}} \mathsf{D}^a_{\{k,...,n\}\setminus(T\cup\{i\}),T\cup\{1,...,k-1,i\}} \oplus \mathsf{D}^c_{S\setminus\{i\},\mathsf{wait}_k(T\cup\{i\}),\mathsf{boundary}_k(T\cup\{i\})}$.

PROOF: The proof is rather similar to that of Lemma 4, but the processes involved are somewhat more complicated.

$$
\begin{array}{rcl}
\mathsf{D}^c_{S,T,k} \,|\, \mathsf{P}_i^{abort} & \equiv & (\nu\vec{c}a)(vote_i[\overline{c_i},\overline{a}] \,|\, \Pi_{j\in S\setminus\{i\}} \mathsf{C}_j^{wait} \,|\, \Pi_{j\in T}\overline{c_j} \,|\, c_k.....c_n.\mathsf{C}_{final}^{commit} \\
& & |\, \mathsf{C}^{or}) \,|\, \Pi_{j\in T\cup\{1,...,k-1\}} dec_j[!\overline{commit_j}, !\overline{abort_j}] \\
& & |\, \overline{vote_i}\mathsf{right} \,|\, !\overline{abort_i} \\
& \approx^{H_{T,i,k-1}} & (\nu\vec{c}a)(\overline{a} \,|\, \Pi_{j\in S\setminus\{i\}} \mathsf{C}_j^{wait} \,|\, c_k.....c_n.\mathsf{C}_{final}^{commit} \,|\, \mathsf{C}^{or}) \\
& & |\, \Pi_{j\in T\cup\{1,...,k-1\}} dec_j[!\overline{commit_j}, !\overline{abort_j}] \,|\, !\overline{abort_i} \\
& \approx_a^{H_{T,i,k-1}} & (\nu\vec{c}a)(\overline{a} \,|\, \Pi_{j\in S\setminus\{i\}} \mathsf{C}_j^{wait} \,|\, \mathsf{C}^{or}) \\
& & |\, \Pi_{j\in T\cup\{1,...,k-1\}} dec_j[!\overline{commit_j}, !\overline{abort_j}] \,|\, !\overline{abort_i} \qquad (3.1) \\
& \approx^{H_{T,i,k-1}} & (\nu\vec{c}a)(\overline{a} \,|\, \mathsf{C}^{or}) \\
& & |\, \Pi_{j\in T\cup\{1,...,k-1\}} dec_j[!\overline{commit_j}, !\overline{abort_j}] \,|\, !\overline{abort_i} \\
& \approx^{H_{T,i,k-1}} & \Pi_{j=1}^n \overline{dec_j}\mathsf{right} \,|\, \Pi_{j\in T\cup\{1,...,k-1\}} dec_j[!\overline{commit_j}, !\overline{abort_j}] \\
& & |\, !\overline{abort_i} \\
& \approx^{H_{T,i,k-1}} & \Pi_{j=k,j\notin T\cup\{i\}}^n \overline{dec_j}\mathsf{right} \,|\, \Pi_{j\in T\cup\{1,...,k-1,i\}} !\overline{abort_i} \\
& \equiv & \Pi_{j\in S\setminus\{i\}}^n \overline{dec_j}\mathsf{right} \,|\, \Pi_{j\in T\cup\{1,...,k-1,i\}} !\overline{abort_i} \\
& \equiv & \mathsf{D}^a_{\{k,...,n\}\setminus(T\cup\{i\}),T\cup\{1,...,k-1,i\}}
\end{array}
$$

(3.1) is a consequence of Lemma 3.3. All other substantial equations in the last derivation are obtained by Lemma 3.1. This shows (1). As to the proof for (2), we reason as follows.

$$
\begin{array}{rcl}
\mathsf{D}^c_{S,T,k} \,|\, \mathsf{P}_i^{commit} & \equiv & (\nu\vec{c}a)(vote_i[\overline{c_i},\overline{a}] \,|\, \Pi_{j\in S\setminus\{i\}} \mathsf{C}_j^{wait} \,|\, \Pi_{j\in T}\overline{c_j} \,|\, c_k.....c_n.\mathsf{C}_{final}^{commit} \\
& & |\, \mathsf{C}^{or}) \,|\, \Pi_{j\in T\cup\{1,...,k-1,i\}} dec_j[!\overline{commit_j}, !\overline{abort_j}] \,|\, \overline{vote_i}\mathsf{left} \\
& \approx^{H_{T,i,k-1}} & (\nu\vec{c}a)(\overline{c_i} \,|\, \Pi_{j\in S\setminus\{i\}} \mathsf{C}_j^{wait} \,|\, \Pi_{j\in T}\overline{c_j} \,|\, c_k.....c_n.\mathsf{C}_{final}^{commit} \\
& & |\, \mathsf{C}^{or}) \,|\, \Pi_{j\in T\cup\{1,...,k-1,i\}} dec_j[!\overline{commit_j}, !\overline{abort_j}]
\end{array}
$$

Now we have 3 possibilities. The first is that $i > k$. Then the equation is easily completed.

$$
\begin{array}{rcl}
... & \equiv & \mathsf{D}^c_{S\setminus\{i\},T\cup\{i\},k} \\
& = & \mathsf{D}^c_{S\setminus\{i\},\mathsf{wait}_k(T\cup\{i\}),\mathsf{boundary}_k(T\cup\{i\})}
\end{array}
$$

The second case is that $i = k$ but $S \setminus \{i\} \neq \emptyset$. Assume the minimal element of $S \setminus \{i\}$ is $k+t \leq n$. Then $k+1, k+2, ..., k+t-1 \in T$, but $k+t \notin T$ and we proceed

$$\begin{array}{rcl}
... & \equiv & (\nu \vec{c}a)(\Pi_{j \in S \setminus \{i\}} \mathsf{C}_j^{wait} \mid \Pi_{j \in T \setminus \{k,...,k+t-1\}} \overline{c_j} \mid \Pi_{j=1}^{t-1} \overline{c_{k+j}} \mid c_k.....c_n.\mathsf{C}_{final}^{commit} \\
 & & \mid \mathsf{C}^{or}) \mid \Pi_{j \in T \cup \{1,...,k-1,i\}} dec_j[!\overline{commit_j}, !\overline{abort_j}] \\
\approx^{H_{T,i,k-1}} & & (\nu \vec{c}a)(\Pi_{j \in S \setminus \{i\}} \mathsf{C}_j^{wait} \mid c_{k+t}.....c_n.\mathsf{C}_{final}^{commit} \\
 & & \mid \mathsf{C}^{or}) \mid \Pi_{j \in T \cup \{1,...,i\}} dec_j[!\overline{commit_j}, !\overline{abort_j}] \\
 & \equiv & \mathsf{D}^c_{S \setminus \{i\}, \mathsf{wait}_k(T \cup \{i\}), \mathsf{boundary}_k(T \cup \{i\})}.
\end{array}$$

The last possibility is that $i = k$ and $T = \{k+1, ..., n\}$. Then

$$\begin{array}{rcl}
... & \equiv & (\nu \vec{c}a)(\Pi_{j \in T \setminus \{k,...,n\}} \overline{c_j} \mid \Pi_{j=k}^{n} \overline{c_j} \mid c_k.....c_n.\mathsf{C}_{final}^{commit} \mid \mathsf{C}^{or}) \\
 & & \mid \Pi_{j=1}^{n} dec_j[!\overline{commit_j}, !\overline{abort_j}] \\
 & \equiv & (\nu \vec{c}a)(\Pi_{j=k}^{n} \overline{c_j} \mid c_k.....c_n.\mathsf{C}_{final}^{commit} \mid \mathsf{C}^{or}) \\
 & & \mid \Pi_{j=1}^{n} dec_j[!\overline{commit_j}, !\overline{abort_j}] \\
\approx^{H_{T,i,k-1}} & & \mathsf{C}_{final}^{commit} \mid \Pi_{j=1}^{n} dec_j[!\overline{commit_j}, !\overline{abort_j}] \\
 & \equiv & \Pi_{j=1}^{n} \overline{dec_j}\mathsf{left} \mid \Pi_{j=1}^{n} dec_j[!\overline{commit_j}, !\overline{abort_j}] \\
\approx^{H_{T,i,k-1}} & & \Pi_{j=1}^{n} !\overline{commit_j} \\
 & \equiv & \mathsf{D}^c_{S \setminus \{i\}, \mathsf{wait}_k(T \cup \{i\}), \mathsf{boundary}_k(T \cup \{i\})}
\end{array}$$

For (3), we note that if $\mathsf{D}^c_{S,T,k} \xrightarrow{l} \mathsf{D}$ is a transition that is not $H_{T,i,k-1}$-hidden, then one of the following must be true.

1. Either $l = vote_j\mathsf{left}$ for some $j \in S$ and $\mathsf{D} \equiv \mathsf{D}^c_{S \setminus \{j\}, \mathsf{wait}_k(T \cup \{j\}), \mathsf{boundary}_k(T \cup \{j\})}$. With $k' = \mathsf{boundary}_k(T \cup \{i\})$ and $T' = \mathsf{wait}_k(T \cup \{i\})$ we then have

$$\begin{array}{rcl}
\mathsf{D}^a_{\{k,...,n\} \setminus (T \cup \{i\}), T \cup \{1,...,k-1,i\}} \oplus \mathsf{D}^c_{S \setminus \{i\}, T', k'} & & \\
 & \xrightarrow{\tau} & \mathsf{D}^c_{S \setminus \{i\}, T', k'} \\
 & \xrightarrow{l} & \mathsf{D}^c_{S \setminus \{i,j\}, \mathsf{wait}_{k'}(T' \cup \{j\}), \mathsf{boundary}_{k'}(T' \cup \{j\})}.
\end{array}$$

2. Alternatively, $l = vote_j\mathsf{right}$ for some $j \in S$ (so in particular, $j \notin \{1, ..., k-1, i\} \cup T$) and $\mathsf{D} \approx_a^{H_{T,i,k-1}} \mathsf{D}^a_{\{k,...,n\} \setminus T, T \cup \{1,...,k-1\}}$. In this case

$$\begin{array}{rcl}
\mathsf{D}^a_{\{k,...,n\} \setminus (T \cup \{i\}), T \cup \{1,...,k-1,i\}} \oplus \mathsf{D}^c_{S \setminus \{i\}, T', k'} & & \\
 & \xrightarrow{\tau} & \mathsf{D}^a_{\{k,...,n\} \setminus (T \cup \{i\}), T \cup \{1,...,k-1,i\}}
\end{array}$$

While (1) is immediate from the syntax, the application of $H_{T,i,k-1}$-expansion (2) is verified by the following derivation.

$$\begin{aligned}\mathsf{D} &\equiv (\nu \vec{c}a)(\Pi_{j\in S}\mathsf{C}_j^{wait} \mid \overline{a} \mid \Pi_{j\in T}\overline{c_j} \mid \mathsf{C}_k^{and} \mid \mathsf{C}^{or}) \\ &\quad \mid \Pi_{i\in T\cup\{1,\ldots,k-1\}} dec_i[!\overline{commit_i}, !\overline{abort_i}] \\ &\approx^{H_{T,i,k-1}} (\nu \vec{c}a)(\Pi_{j\in S}\mathsf{C}_j^{wait} \mid \Pi_{j\in T}\overline{c_j} \mid \mathsf{C}_k^{and} \mid \Pi_{i=1}^{n}\overline{dec_i}\mathsf{right}) \\ &\quad \mid \Pi_{i\in T\cup\{1,\ldots,k-1\}} dec_i[!\overline{commit_i}, !\overline{abort_i}]\end{aligned}$$

By Lemma 3.1.

$$\ldots \approx_a^{H_{T,i,k-1}} \Pi_{i=1}^{n}\overline{dec_i}\mathsf{right} \mid \Pi_{i\in T\cup\{1,\ldots,k-1\}} dec_i[!\overline{commit_i}, !\overline{abort_i}]$$

Now we apply Lemma 3.3.

$$\ldots \approx^{H_{T,i,k-1}} \Pi_{i=k,i\notin T}^{n}\overline{dec_i}\mathsf{right} \mid \Pi_{i\in T\cup\{1,\ldots,k-1\}}!\overline{abort_i}$$

Finally Lemma 3.1 comes to our aid.

$$\ldots \equiv \mathsf{D}^a_{\{k,\ldots,n\}\setminus T, T\cup\{1,\ldots,k-1\}}$$

Now we apply Theorem 16.1. □

We can now combine Lemmas 4 and 5.

PROPOSITION 1 $\mathsf{C} \mid \Pi_{i=1}^{n}\mathsf{P}_i \approx^{H_n} \mathsf{Abort} \oplus \mathsf{Commit}$

PROOF: From Lemma 5 we know that

$$\mathsf{D}^c_{\{1,\ldots,n\},\emptyset,1} \mid \mathsf{P}_1 \approx^{H_{\{1\}}} \mathsf{D}^c_{\{2,\ldots,n\},\emptyset,2} \oplus \mathsf{D}^a_{\{2,\ldots,n\}\{1\}}.$$

Now

$$\begin{array}{lll}\mathsf{D}^c_{\{2,\ldots,n\},\emptyset,2} \mid \mathsf{P}_2 & \approx^{H_{\{1,2\}}} & \mathsf{D}^c_{\{3,\ldots,n\},\emptyset,3} \oplus \mathsf{D}^a_{\{3,\ldots,n\}\{1,2\}} \\ \mathsf{D}^a_{\{2,\ldots,n\},\{1\}} \mid \mathsf{P}_2 & \approx^{H_{\{2\}}} & \mathsf{D}^a_{\{3,\ldots,n\}\{1,2\}}.\end{array}$$

This together with Lemmas 4 and 5 means we can apply Theorem 16.3 to obtain

$$\mathsf{D}^c_{\{1,\ldots,n\},\emptyset,1} \mid \mathsf{P}_1 \mid \mathsf{P}_2 \approx^{H_{\{1,2\}}} \mathsf{D}^c_{\{3,\ldots,n\},\emptyset,3} \oplus \mathsf{D}^a_{\{3,\ldots,n\}\{1,2\}}.$$

Continuing this way, we eventually get

$$\begin{array}{lll}\mathsf{D}^c_{\{1,\ldots,n\},\emptyset,1} \mid \mathsf{P}_1 \mid \ldots \mid \mathsf{P}_n & \approx^{H_{\{1,\ldots,n\}}} & \mathsf{D}^c_{\emptyset,\emptyset,n+1} \oplus \mathsf{D}^a_{\{1,\ldots,n\}} \\ & \equiv & \mathsf{Commit} \oplus \mathsf{Abort},\end{array}$$

as required. □

In the light of Theorem 15.1, Theorem 12 is an immediate consequence of Proposition 1 while Theorem 13 follows from Lemmas 4 and 5.

3.4 What Lies Ahead?

We now abandon the benign world of error free interaction and create several new and scary ones. We begin by adding time and timers. On its own that is insufficient for reaching the shores of DS, so we will also add sites and message failures in Chapter 5. The resulting calculus is expressive enough for a "proper" 2PCP where things can really go wrong and the recovery mechanisms of the protocol come into their own. Nevertheless, the correctness proof just outlined remains largely functional in this new setting, except for a proliferation of (unobservable) intermediate states. Things become more challenging once we add process failure and recovery in Chapter 7. We can still use Divide-and-Conquer, but less frequently, because the equational steps will be much coarser.

Chapter 4

Timers

This chapter extends the asynchronous π-calculus with a timer construct and studies some of the resulting equational properties.

4.1 Introduction

Here comes the first extension: time and their manipulation by way of timers. The rationale for inclusion of timers is their ubiquity and indispensability in distributed computation, where they are used mainly for the efficient implementation of periodic behaviour and error recovery. An instance of the latter is waiting for the predetermined amount of time for an event to happen, for example waiting for the arrival of an acknowledge packet in a network flow control algorithm or for commitment of a transaction. An example of periodic behaviour is the time-slicing that many preemptive operating systems employ to arbitrate between the competing demands of processes and threads for CPU usage. But why use timers? CPUs have tightly constrained temporal properties which could be used to implement cyclic or waiting behaviour, for example by "busy waiting". This usually means for the CPU to execute a loop a predetermined number of times. Although clearly possible, there is a drawback. While looping, the CPU cannot be used for other, more useful computations, at least not without great effort. This is unacceptable, in particular for OS schedulers, the quality of which is partly measured by just how much CPU cycles it manages to assign to processes and threads.

This is where timers help. They are specialised time-keeping hardware (= clocks) which is optimised for efficiently carrying out three jobs with minimal burden for the CPU.

- Timers can be *set* or *started*. That means that the CPU requests to be notified as exactly as possible, when a specified number of atomic time steps have passed, measured from when the timer was started.

- Timers can be *stopped*. In other words, the CPU can communicate to the timer that it is no longer interested in being informed about the passing of some number of time units.

- Timers can *time-out*, i.e. the CPU is informed, as requested, that a certain number of time units have passed. This allows the CPU to execute some predetermined code fragment to deal with this situation.

The advantage timers have over busy waiting lies in the the fact that these three tasks are easy to realise physically and without burdening other computations, in contrast to the implementation of the mechanism allowing busy-waiting, which essentially hogs the CPU.

There are many ways of incorporating timers into π-calculi and we will survey some but by no means all design alternatives in §4.8. Our proposal below seeks to meet the following objectives

- The model should capture as accurately as possible the intuitive behaviour of timers, as sketched above. Its way of formalising time-passing and how that relates to computation should reflect the expectations of well-educated programmers. In particular, paradoxical temporal behaviour, for example the possibility of executing an infinite number of computational steps in finite time, should be avoided.

- We would like to be able to express all temporal behaviour that can be algorithmically described using conventional computer systems and languages. In particular, OS schedulers and the TCP error recovery mechanism should be easy to model. As we will point out at the end of this chapter, some previous calculi seem to have problems in this area.

- As much as possible of the existing π-calculus theory should be retained. In particular, we wish to retain the asynchronous π-calculus as a sub-calculus and we would like to be able to use name passing synchronisation trees and the associated notions of equivalence as key semantic tools. This requirement rules out many previous approaches to timed process calculi that use time-passing actions to broadcast the flow of time.

- Our calculus is emphatically not intended to just be able to model this or that finite state protocol. Instead we would ultimately like to precisely and concisely model realistic programming languages such as Java or C++ which have timer constructs. It is likely that this would require extensions beyond adding timers, but starting from non-compositional formalisms such as CCS or CSP would probably stop our ambition in its tracks.

It seems that many of the details of how timers are formalised do not matter, in the sense that many but not all alternative formulations can encode each other. The key problem with adding timers is that the resulting equational theory is drastically different from its untimed counterpart. Many of the tools of untimed process theory no longer work as conveniently as in their original setting.

Before presenting our timed extension of the π-calculus, it may be deemed appropriate to clarify what we mean by "time". Unfortunately, to this day, no truly convincing account of the nature of time has been given, or if it has, it has not been communicated to the present author. Instead of philosophical speculation, we simply pretend we understand the concept well enough to be able to carry out our programme. In particular, we will boldly assume without presenting any evidence, that infinite linear orders such as $\mathbb{N}$, $\mathbb{Q}$ or $\mathbb{R}$ are sufficiently good mathematical models of time for our purpose.

4.2 The Calculus

We incorporate timers as a simple syntactic extension. Timers are processes of the form

$$\mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q}).$$

Here $t > 0$ is an integer determining the number of timesteps before the timer *times out*. Q is the *time-out continuation*. It will be launched when the timer times out without having been stopped. The *time-in continuation* is $x(\vec{v}).\mathsf{P}$. By interaction at x it is possible to *stop* the timer, as long as it has not already timed-out. By stopping the timer, we force it to become $\mathsf{P}\{\vec{y}/\vec{v}\}$, where $\vec{y}$ is the tuple of names that has been passed by the stopping interaction at x.

In summary, here is the syntax of the resulting *timed asynchronous π-calculus* π_t.

P	$::=$	$x(\vec{y}).\mathsf{P}$	input prefix
	$\mid$	$\overline{x}\langle\vec{y}\rangle$	output
	$\mid$	$\mathsf{P}\mid\mathsf{Q}$	parallel composition
	$\mid$	$(\nu x)\mathsf{P}$	hiding
	$\mid$	$\mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q})$	timer
	$\mid$	$!x(\vec{v}).\mathsf{P}$	lazy replication
	$\mid$	$\mathsf{0}$	inaction

Clearly the original asynchronous π-calculus is a subcalculus of π_t. The reason for choosing lazy replication over the unrestricted replication of Chapter 2 will be explained in §4.3.1.

We must emphasise that timers like ours are hardly novel. Many other timed calculi have similar constructs. It is just that the effect of the introduction of timers

to π-calculi has not been studied. The main point of the present chapter is to fill this gap.

4.3 Semantics

Next we describe how computation proceeds in our extended calculus. The key questions are: what is the atomic unit of time and how is time passing communicated to the timers?

The answer to the first question is that we count each computational step, that is each name passing interaction, as taking one atomic step in time. This is pretty much the only choice in π-calculi, which try to make things as simple as possible: nothing apart from name-passing takes place, computation is just discrete sequences of interactions. Of course this design decision prevents having elements of $\mathbb{R}$ as durations. But while reals might be appealing from the point of view of most physical models of space-time, conventional computers just have not got access to arbitrarily precise durations, only to integer multiples of the atomic granularity of time measurement, often milli- or microseconds. Our choice reflects this limitation.

The second question allows for more variation in its answer. Many models of timed computation use specifically designated time-passing actions to communicate the flow of time to all time-sensitive processes. Following this tradition would have prevented recycling name-passing synchronisation trees as a semantic model, in violation of our design goals. To avoid this sacrifice, our technical development of timers is instead based on *implicitly* communicating the passing of time with the help of the following *timestepper function* ϕ, which acts on processes.

$$\phi(\mathsf{P}) = \begin{cases} \mathsf{timer}^{t-1}(\mathsf{Q},\mathsf{R}) & \mathsf{P} = \mathsf{timer}^{t}(\mathsf{Q},\mathsf{R}), t > 1 \\ \mathsf{R} & \mathsf{P} = \mathsf{timer}^{t}(\mathsf{Q},\mathsf{R}), t \leq 1 \\ \phi(\mathsf{Q})|\phi(\mathsf{R}) & \mathsf{P} = \mathsf{Q}|\mathsf{R} \\ (\nu x)\phi(\mathsf{Q}) & \mathsf{P} = (\nu x)\mathsf{Q} \\ \mathsf{P} & \text{otherwise.} \end{cases}$$

Thus $\phi(\mathsf{P})$ ticks each timer in P by one discrete unit: this can be thought of as the passing of, say, one micro-second in a global clock. Note that this function only acts on non-guarded timers: it does not influence timers under prefixes. This essentially indicates that a timer starts only after the guarding prefix is taken off, i.e. after the process is launched into the environment. Timers guarded by prefixes are said to be *inactive*, otherwise they are *active*. A consequence of this choice of timestepper is that timers under a replication operator are inactive.

$$
\begin{array}{lcllcl}
\mathsf{fn}(\mathsf{timer}^t(\mathsf{P},\mathsf{Q})) & = & \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q}) & \mathsf{bn}(\mathsf{timer}^t(\mathsf{P},\mathsf{Q})) & = & \mathsf{bn}(\mathsf{P}) \cup \mathsf{bn}(\mathsf{Q}) \\
\mathsf{fn}(\overline{x}\langle\vec{y}\rangle) & = & \{x, \vec{y}\} & \mathsf{bn}(\overline{x}\langle\vec{y}\rangle) & = & \emptyset \\
\mathsf{fn}(x(\vec{y}).\mathsf{P}) & = & \{x\} \cup (\mathsf{fn}(\mathsf{P}) \setminus \{\vec{y}\}) & \mathsf{bn}(x(\vec{y}).\mathsf{P}) & = & \mathsf{bn}(\mathsf{P}) \cup \{\vec{y}\} \\
\mathsf{fn}(!x(\vec{v}).\mathsf{P}) & = & \{x\} \cup (\mathsf{fn}(\mathsf{P}) \setminus \{\vec{y}\}) & \mathsf{bn}(!x(\vec{v}).\mathsf{P}) & = & \{x\} \cup (\mathsf{fn}(\mathsf{P}) \setminus \{\vec{y}\}) \\
\mathsf{fn}(\mathsf{P}|\mathsf{Q}) & = & \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q}) & \mathsf{bn}(\mathsf{P}|\mathsf{Q}) & = & \mathsf{bn}(\mathsf{P}) \cup \mathsf{bn}(\mathsf{Q}) \\
\mathsf{fn}((\nu x)\mathsf{P}) & = & \mathsf{fn}(\mathsf{P}) \setminus \{x\} & \mathsf{bn}((\nu x)\mathsf{P}) & = & \mathsf{bn}(\mathsf{P}) \cup \{x\} \\
\mathsf{fn}(0) & = & \emptyset & \mathsf{bn}(0) & = & \emptyset
\end{array}
$$

$\equiv$ is the least congruence satisfying the following rules.

$$
\begin{array}{ll}
\mathsf{P} \equiv_\alpha \mathsf{Q} \Rightarrow \mathsf{P} \equiv \mathsf{Q} & (\nu x)0 \equiv 0 \\
\mathsf{P}|\mathsf{Q} \equiv \mathsf{Q}|\mathsf{P} & \mathsf{P}|0 \equiv \mathsf{P} \\
\mathsf{P}|(\mathsf{Q}|\mathsf{R}) \equiv (\mathsf{P}|\mathsf{Q})|\mathsf{R} & x \notin \mathsf{fn}(\mathsf{P}) \Rightarrow \mathsf{P}|(\nu x)\mathsf{Q} \equiv (\nu x)(\mathsf{P}|\mathsf{Q}) \\
(\nu x)(\nu y)\mathsf{P} \equiv (\nu y)(\nu x)\mathsf{P}
\end{array}
$$

$$
\begin{array}{rl}
(\textsc{Com}) & \dfrac{}{x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \;\rightarrow\; \mathsf{P}\{\vec{v}/\vec{y}\}} \\[2ex]
(\textsc{Rep}) & \dfrac{}{!x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \rightarrow !x(\vec{v}).\mathsf{P} \mid \mathsf{P}\{\vec{v}/\vec{y}\}} \\[2ex]
(\textsc{TimeIn}) & \dfrac{}{\mathsf{timer}^{t+1}(x(\vec{v}).\mathsf{P},\mathsf{Q}) \mid \overline{x}\langle\vec{y}\rangle \rightarrow \mathsf{P}\{\vec{y}/\vec{v}\}} \\[2ex]
(\textsc{Par}) & \dfrac{\mathsf{P} \rightarrow \mathsf{P}'}{\mathsf{P}|\mathsf{Q} \rightarrow \mathsf{P}'|\phi(\mathsf{Q})} \\[2ex]
(\textsc{Res}) & \dfrac{\mathsf{P} \rightarrow \mathsf{Q}}{(\nu x)\mathsf{P} \rightarrow (\nu x)\mathsf{Q}} \\[2ex]
(\textsc{Idle}) & \dfrac{}{\mathsf{P} \rightarrow \phi(\mathsf{P})} \\[2ex]
(\textsc{Cong}) & \dfrac{\mathsf{P} \equiv \mathsf{P}' \quad \mathsf{P}' \rightarrow \mathsf{Q}' \quad \mathsf{Q}' \equiv \mathsf{Q}}{\mathsf{P} \rightarrow \mathsf{Q}}
\end{array}
$$

Figure 4.1: The inductive definitions of the dynamics of the timed asynchronous π-calculus.

4.3.1 Dynamics and Structural Congruence

The semantics of timers, summarised in Figure 4.1, requires just two new reduction rules and some modification of one rule of the asynchronous π-calculus. As each interaction ticks all active timers by one unit, (Par) must now be

$$(\text{Par}) \quad \frac{\mathsf{P} \to \mathsf{P}'}{\mathsf{P}|\mathsf{Q} \to \mathsf{P}'|\phi(\mathsf{Q})}$$

plus a symmetric variant. Clearly, when restricted to timer-free processes, the new rule works exactly as the original rule in §2.3.1. The first entirely new rule describes how a timer can be stopped.

$$(\text{TimeIn}) \quad \frac{}{\mathsf{timer}^{t+1}(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \,|\, \overline{x}\langle\vec{y}\rangle \to \mathsf{P}\{\vec{y}/\vec{v}\}}$$

(Par) and (TimeIn) are probably inevitable, once one accepts our concept of timer with its associated constraints. The last new rule is more controversial.

$$(\text{Idle}) \quad \frac{}{\mathsf{P} \to \phi(\mathsf{P})}$$

It allows the computation to pause or *idle* at arbitrary moments and, through repeated application, for an unlimited period of time. Of course some idling is unavoidable for timers to be efficient, for example for allowing the computation to recover from errors such as deadlocks or lost messages, which would otherwise stall it forever. Nevertheless, other formalisms are more restrictive and make progress assumptions: they may for instance require that idling is permitted only if no interaction is possible at all (this is the *maximal progress assumption*). We have decided to leave all progress assumptions out of the basic semantics.

- They would compromise the pleasing simplicity of our extensions.
- They would diminish the flexibility of our approach which allows to model systems that do not give progress guarantees. For example, many operating systems might suspend processes at unpredictable moments for arbitrary amounts of time.
- We are not losing expressive power, as arbitrary progress requirements can be modelled on top of the unconstrained model, as described briefly in §4.7.2.
- Finally, it is not clear to us that it is always computably feasible to decide, or at least decide efficiently, whether a process is blocked and must idle or not.

This makes implementing progress assumptions problematic.

Lazy Replication

We can now see why lazy replication is a good idea in the presence of timers: if we allowed unrestricted replication, the process

$$\begin{aligned} !\mathsf{timer}^t(\mathsf{P},\mathsf{Q}) &\equiv \mathsf{timer}^t(\mathsf{P},\mathsf{Q})|!\mathsf{timer}^t(\mathsf{P},\mathsf{Q}) \\ &\equiv \mathsf{timer}^t(\mathsf{P},\mathsf{Q})|\mathsf{timer}^t(\mathsf{P},\mathsf{Q})|!\mathsf{timer}^t(\mathsf{P},\mathsf{Q}) \\ &\equiv \ldots \end{aligned}$$

could be formed and by mere application of the (Cong) rule one could start any number of timers. This is not clean. We want to start a timer exactly when its guarding input prefix is removed by interaction and we want to start at most one timer per interaction. An alternative approach would be to consider timers under a replication (but no input prefix) active, but that would cause all manner of problems

Free Input and Output Names

For some theorems later, it will be useful to be able to speak of *free input names* and *free output names*. The former are those free names that are used as input subjects without being bound.

$$\begin{aligned} \mathsf{fin}(\overline{x}\langle\vec{y}\rangle) &= \emptyset \\ \mathsf{fin}(x(\vec{v}).\mathsf{P}) &= (\mathsf{fin}(\mathsf{P}) \setminus \{\vec{v}\}) \cup \{x\} \\ \mathsf{fin}(!x(\vec{v}).\mathsf{P}) &= (\mathsf{fin}(\mathsf{P}) \setminus \{\vec{v}\}) \cup \{x\} \\ \mathsf{fin}(\mathsf{timer}^t(x(\vec{v}).\mathsf{P},\mathsf{Q})) &= (\mathsf{fin}(\mathsf{P}) \setminus \{\vec{v}\}) \cup \{x\} \cup \mathsf{fin}(\mathsf{Q}) \\ \mathsf{fin}((\nu x)\mathsf{P}) &= \mathsf{fin}(\mathsf{P}) \setminus \{x\}, \\ \mathsf{fin}(\mathsf{P}|\mathsf{Q}) &= \mathsf{fin}(\mathsf{P}) \cup \mathsf{fin}(\mathsf{Q}) \\ \mathsf{fin}(0) &= \emptyset \end{aligned}$$

Free output names enjoy a symmetric definition.

$$\begin{aligned} \mathsf{fon}(\overline{x}\langle\vec{y}\rangle) &= \{x\} \\ \mathsf{fon}(x(\vec{v}).\mathsf{P}) &= \mathsf{fon}(\mathsf{P}) \setminus \{\vec{v}\} \\ \mathsf{fon}(!x(\vec{v}).\mathsf{P}) &= \mathsf{fon}(\mathsf{P}) \setminus \{\vec{v}\} \\ \mathsf{fon}(\mathsf{timer}^t(x(\vec{v}).\mathsf{P},\mathsf{Q})) &= (\mathsf{fon}(\mathsf{P}) \setminus \{\vec{v}\}) \cup \mathsf{fon}(\mathsf{Q}) \\ \mathsf{fon}((\nu x)\mathsf{P}) &= \mathsf{fon}(\mathsf{P}) \setminus \{x\}, \\ \mathsf{fon}(\mathsf{P}|\mathsf{Q}) &= \mathsf{fon}(\mathsf{P}) \cup \mathsf{fon}(\mathsf{Q}) \\ \mathsf{fon}(0) &= \emptyset \end{aligned}$$

Active Names

In addition to free and bound names, our later development also makes use of *active names*. They are those names that can be used immediately for interaction.

DEFINITION 15 Now $\mathsf{an}(\mathsf{P})$, the *active names* of P are given by induction on the syntax of P:

$$\begin{array}{lcl}
\mathsf{an}(\overline{x}\langle\vec{y}\rangle) & = & \{x\} \\
\mathsf{an}(x(\vec{v}).\mathsf{P}) & = & \{x\} \\
\mathsf{an}(!x(\vec{v}).\mathsf{P}) & = & \{x\} \\
\mathsf{an}(\mathsf{timer}^t(x(\vec{v}).\mathsf{P},\mathsf{Q})) & = & \{x\} \\
\mathsf{an}((\nu x)\mathsf{P}) & = & \mathsf{an}(\mathsf{P}) \setminus \{x\}, \\
\mathsf{an}(\mathsf{P}|\mathsf{Q}) & = & \mathsf{an}(\mathsf{P}) \cup \mathsf{an}(\mathsf{Q}) \\
\mathsf{an}(0) & = & \emptyset.
\end{array}$$

Contexts

DEFINITION 16 The definition of the set of *contexts* extends that in Chapter 2 with

$$C[\cdot] \quad ::= \quad ... \;\mid\; \mathsf{timer}^t(x(\vec{v}).C[\cdot],\mathsf{P}) \;\mid\; \mathsf{timer}^t(x(\vec{v}).\mathsf{P},C[\cdot]).$$

The definition of *linear contexts* and *reduction contexts* remain syntactically unchanged because neither the time-in continuation nor the time-out continuation are active in a timer.

It is often useful to be able to say that a context makes a reduction step, as for example $C[\cdot] = \overline{x}|x|y.[\cdot]$ which can reduce to $C'[\cdot] = y.[\cdot]$, because no matter what process P we choose, we can always derive $C[\mathsf{P}]{\rightarrow}C'[\mathsf{P}]$. The next definition clarifies this and related matters.

DEFINITION 17 If $C[\mathsf{P}]{\rightarrow}C'[\mathsf{P}]$ for all P, then we write $C[\cdot]{\rightarrow}C'[\cdot]$. This generalises to $C[\cdot] \twoheadrightarrow C'[\cdot]$ as expected and we say $C[\cdot]$ *generically reduces* to $C'[\cdot]$. $C[\cdot]\downarrow_x$ is defined similarly. We define $\phi(C[\cdot])$ inductively, up to $\equiv$.

$$\phi(C[\cdot]) = \begin{cases}
[\cdot] & C[\cdot] = [\cdot] \\
\phi(C')[\cdot]\,|\,\phi(\mathsf{P}) & C[\cdot] = C'[\cdot]\,|\,\mathsf{P} \\
(\nu x)\phi(C')[\cdot] & C[\cdot] = (\nu x)C'[\cdot] \\
C'[\cdot] & C[\cdot] = \mathsf{timer}^1(\mathsf{P},C'[\cdot]) \\
\mathsf{timer}^t(\mathsf{P},C'[\cdot]) & C[\cdot] = \mathsf{timer}^{t+1}(\mathsf{P},C'[\cdot]) \\
\mathsf{Q} & C[\cdot] = \mathsf{timer}^1(x(\vec{v}).C'[\cdot],\mathsf{Q}) \\
\mathsf{timer}^t(x(\vec{v}).C'[\cdot],\mathsf{Q}) & C[\cdot] = \mathsf{timer}^{t+1}(x(\vec{v}).C'[\cdot],\mathsf{Q}) \\
C[\cdot] & \text{otherwise}
\end{cases}$$

Note that linear context, i.e. what we sloppily call contexts, are not closed under timesteps.

LEMMA 6 *If $C[\cdot]$ is a reduction-context, then $\phi(C[\mathsf{P}]) = \phi(C)[\phi(\mathsf{P})]$. Otherwise $\phi(C[\mathsf{P}]) = \phi(C)[\mathsf{P}]$. If $C[\cdot] \rightarrow C'[\cdot]$ and $C[\cdot]$ is a reduction-context, then $C[\mathsf{P}] \rightarrow C'[\phi(\mathsf{P})]$. Otherwise $C[\mathsf{P}] \rightarrow C'[\mathsf{P}]$.*

PROOF: Straightforward from the definitions. □

Barbs

As we have pointed out in Chapter 2, barbs are one tool to get equivalences. Unfortunately, it is not clear what barbs for timed calculi should look like. Following the train of though that lead Milner and Sangiorgi [79] towards their now classic notion of barb, one could argue as follows: timers do not add active names that can be used for immediate output, hence the definition of strong and weak barbs remains unchanged from §2.3.2. We reiterate it for convenient reference.

DEFINITION 18 The *strong barb* $\downarrow_x$ for the timed π-Calculus is defined by the following rules.

$$\frac{}{\overline{x}\langle\vec{y}\rangle \downarrow_x} \qquad \frac{\mathsf{P} \downarrow_x}{\mathsf{P}|\mathsf{Q} \downarrow_x} \qquad \frac{\mathsf{P} \downarrow_x \quad x \neq y}{(\nu y)\mathsf{P} \downarrow_x}$$

The corresponding *(weak) barb* $\Downarrow_x$ is also derived as in Chapter 2.

$$\mathsf{P} \Downarrow_x \qquad \text{iff} \qquad \mathsf{P} \twoheadrightarrow \mathsf{Q} \downarrow_x$$

We sometimes write $\mathsf{P} \downarrow_{xy}$ to mean $\mathsf{P} \downarrow_x$ and $\mathsf{P} \downarrow_y$.

We state some useful facts about barbs.

LEMMA 7 *1. If $\mathsf{P} \equiv \mathsf{Q}$ and $\mathsf{P} \downarrow_x$ then $\mathsf{Q} \downarrow_x$.*

2. $\mathsf{P} \downarrow_x$ if and only if P is of the form $(\nu\vec{a})(\overline{x}\langle\vec{y}\rangle \mid \mathsf{Q})$ where $x \notin \{\vec{a}\}$.

PROOF: (1) is by induction on the derivation of barbs and on the derivation of $\mathsf{P} \equiv \mathsf{Q}$, while (2) can be proven by induction on the derivation of barbs. □

It will turn out that this is exactly the right notion of barb.

Barbed Congruence

We can now define the standard barbed congruences.

DEFINITION 19 A binary relation $\mathcal{R}$ on processes is a *π_t-congruence* if it it an equivalence, if $\equiv \subseteq \mathcal{R}$ and if $\mathsf{P} \ \mathcal{R} \ \mathsf{Q}$ implies $C[\mathsf{P}] \ \mathcal{R} \ C[\mathsf{Q}]$ for all contexts $C[\cdot]$.

DEFINITION 20 A symmetric binary relation $\mathcal{R}$ on processes is a *strong barbed bisimulation* if it is a π_t-congruence and if $\mathsf{P} \ \mathcal{R} \ \mathsf{Q}$ implies the following.

- For all names x: $\mathsf{P}\downarrow_x$ implies $\mathsf{Q}\downarrow_x$.
- Whenever $\mathsf{P}\rightarrow\mathsf{P}'$ then there is a process Q' such that $\mathsf{Q}\rightarrow\mathsf{Q}'$ and $\mathsf{P}'\ \mathcal{R}\ \mathsf{Q}'$.

It is easy to see that strong barbed bisimulations are closed under arbitrary unions. The largest strong barbed bisimulation is called *strong reduction congruence*. We denote it by $\overset{rc}{\sim}$.

The corresponding notions of *barbed bisimulation* and *reduction congruence* $\overset{rc}{\approx}$ are derived as explained in Chapter 2, by replacing $\downarrow_x$ with $\Downarrow_x$ and $\rightarrow$ with $\twoheadrightarrow$.

In contrast with untimed π-calculi, it will turn out that the addition of timers forces $\overset{rc}{\approx}$ and $\overset{rc}{\sim}$ to coincide.

DEFINITION 21 A binary relation $\mathcal{R}$ on processes is *time-closed* if $(\mathsf{P},\mathsf{Q})\in\mathcal{R}$ implies $(\phi(\mathsf{P}),\phi(\mathsf{Q}))\in\mathcal{R}$.

We will later show that $\overset{rc}{\approx}$ and $\overset{rc}{\sim}$ are time-closed.

Before continuing with the transitional semantics, we present one example of a strong barbed bisimilarity, which will play a role later.

EXAMPLE 1 For all processes P:

$$\mathsf{P}\overset{rc}{\sim}\mathsf{P}\mid(\nu x)\overline{x}.$$

To see that this is the case, define

$$\mathcal{R}=\{(C[\mathsf{P}],C[\mathsf{P}\mid(\nu x)\overline{x}])\mid\mathsf{P}\text{ is a process},C[\cdot]\text{ is a context}\}\cup\equiv$$

It is easy to see that $\mathcal{R}$ preserves strong barbs and is strongly reduction-closed.

4.3.2 Transitional Semantics

Figure 4.2 inductively defines the *synchronous labelled transitions* for π_t, as a first attempt at providing tools for efficient reasoning about (strong) reduction congruence. Transitions are derived from those for the asynchronous π-calculi in the same way that we derived reductions for π_t from those for π_a, by modification of (PAR) and addition of a rule for stopping a timer and one for idling.

THEOREM 17 *Transition and reduction semantics coincide, that is:* $\rightarrow\ =\ \overset{\tau}{\rightarrow}\equiv$.

PROOF: Using an intermediate labelled transition system along the lines of [75]. □

$$\text{(Out)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle} \mathsf{0}}$$

$$\text{(In)}\quad \frac{}{x(\vec{v}).\mathsf{P} \xrightarrow{x(\vec{z})} \mathsf{P}\{\vec{z}/\vec{v}\}}$$

$$\text{(Rep)}\quad \frac{}{!x(\vec{v}).\mathsf{P} \xrightarrow{x(\vec{z})} \mathsf{P}\{\vec{z}/\vec{v}\} \mid !x(\vec{v}).\mathsf{P}}$$

$$\text{(TimeIn)}\quad \frac{}{\mathsf{timer}^{t+1}(x(\vec{v}).\mathsf{P},\mathsf{Q}) \xrightarrow{x(\vec{z})} \mathsf{P}\{\vec{z}/\vec{v}\}}$$

$$\text{(Par)}\quad \frac{\mathsf{P} \xrightarrow{l} \mathsf{P}' \quad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P}|\mathsf{Q} \xrightarrow{l} \mathsf{P}'|\phi(\mathsf{Q})}$$

$$\text{(Com)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{x(\vec{z})} \mathsf{Q}' \quad \{\vec{y}\} \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{\tau} (\nu\vec{y})(\mathsf{P}' \mid \mathsf{Q}')}$$

$$\text{(Res)}\quad \frac{\mathsf{P} \xrightarrow{l} \mathsf{Q} \quad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)}{(\nu x)\mathsf{P} \xrightarrow{l} (\nu x)\mathsf{Q}}$$

$$\text{(Open)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{Q} \quad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle} \mathsf{Q}}$$

$$\text{(Idle)}\quad \frac{}{\mathsf{P} \xrightarrow{\tau} \phi(\mathsf{P})}$$

$$\text{(Alpha)}\quad \frac{\mathsf{P}' \xrightarrow{l} \mathsf{Q} \quad \mathsf{P} \equiv_\alpha \mathsf{P}'}{\mathsf{P} \xrightarrow{l} \mathsf{Q}}$$

Figure 4.2: The standard synchronous transitions for the asynchronous timed π-calculus.

Basic Facts

The next lemma lists various basic facts about π_t that will be used later.

LEMMA 8 *1.* P *contains active timers if and only if* $\mathsf{P} \neq \phi(\mathsf{P})$.

2. If $\mathsf{P} \twoheadrightarrow \mathsf{Q}$ *then* $\mathsf{fn}(\mathsf{Q}) \subseteq \mathsf{fn}(\mathsf{P})$.

3. Assume that $\mathsf{P} \bowtie \mathsf{Q}$ *and* $\mathsf{P}|\mathsf{Q} \twoheadrightarrow_n \mathsf{R}$. *Then there must be processes* Q', R' *such that* $\mathsf{P} \equiv \mathsf{Q}'|\mathsf{R}'$, $\mathsf{P} \twoheadrightarrow_n \mathsf{P}'$, $\mathsf{Q} \twoheadrightarrow_n \mathsf{Q}'$ *and* $\mathsf{P}' \bowtie \mathsf{Q}'$.

4. $\mathsf{P}\downarrow_x$ *iff* $\mathsf{P} \equiv \mathsf{P}' \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{Q}$ *for some appropriate* $\vec{y}$, $\vec{z}$, P' *and* Q.

5. Let $\mathsf{fn}(P) \subseteq \{\vec{a}\}$ *and* $C[\cdot] = (\nu\vec{a})([\cdot] \,|\, \mathsf{timer}^1(x(\vec{v}).\overline{y}, 0))$ *where* b *is fresh. Then* $C[\mathsf{P}] \rightarrow \mathsf{P}\downarrow_b$ *iff* $\mathsf{P}\downarrow_x$.

6. If $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{Q}$ *then* $\{\vec{y}\} \subseteq \{\vec{z}\}$.

7. $\phi(\mathsf{P}\{x/v\}) = \phi(\mathsf{P})\{x/v\}$.

8. If $\mathsf{P} \equiv \mathsf{Q}$ *then* $\phi(\mathsf{P}) \equiv \phi(\mathsf{Q})$.

9. Every strong barbed bisimulation is a barbed bisimulation.

PROOF: All are straightforward by appropriate inductions, sometimes nested. □

Examples

Before moving on, let's consider a few π_t processes and their reductions. The following process implements a *delay operator*:

$$\mathsf{delay}^t(\mathsf{P}) = (\nu x)\mathsf{timer}^t(x.0, \mathsf{P})$$

where $x \notin \mathsf{fn}(\mathsf{P})$. For t units of time, it cannot interact at all, it behaves like 0, but then it evolves into P. It is comparable to the `sleep` operator in Java or POSIX compliant languages. It can be used to implement cyclic behaviour: $(\nu x)(\overline{x} \,|\, !x.\mathsf{delay}^t(\mathsf{P} \,|\, \overline{x}))$ $(x \notin \mathsf{fn}(\mathsf{P}))$ which can spawn P every $t+1$ units of time. The delay operator will be crucial in the proof of Theorem 25. Here is an example

reduction sequence that shows how the delay operator works

$$\begin{aligned}
(\nu x)(\overline{x} \,|\, x.\overline{y}) \,|\, \mathsf{delay}^3(y.\overline{a}) &= (\nu x)(\overline{x} \,|\, x.\overline{y}) \,|\, (\nu x)\mathsf{timer}^3(x.0, y.\overline{a}) \\
&\to \overline{y} \,|\, \phi((\nu x)\mathsf{timer}^3(x.0, y.\overline{a})) \\
&= \overline{y} \,|\, (\nu x)\mathsf{timer}^2(x.0, y.\overline{a}) \\
&\to \overline{y} \,|\, \phi((\nu x)\mathsf{timer}^2(x.0, y.\overline{a})) \\
&= \overline{y} \,|\, (\nu x)\mathsf{timer}^1(x.0, y.\overline{a}) \\
&\to \overline{y} \,|\, \phi((\nu x)\mathsf{timer}^1(x.0, y.\overline{a})) \\
&= \overline{y} \,|\, y.\overline{a} \\
&\to \overline{a} \\
&\to \overline{a} \\
&\to \ldots
\end{aligned}$$

The first timestep is induced by the interaction that launches $\overline{y}$. But once that has happened there are not further interactions, that could tick the clock. Hence we have to rely on (IDLE) for the next two reductions. When the process has reduced to become $\overline{a}$, we can again only use (IDLE).

The second example uses delay operators to obtain cyclic behaviour.

$$\begin{aligned}
(\nu x)(\overline{x} \,|\, !x.\mathsf{delay}^t(\mathsf{P} \,|\, \overline{x})) &\to (\nu x)(!x.\mathsf{delay}^t(\mathsf{P} \,|\, \overline{x}) \,|\, \mathsf{delay}^t(\mathsf{P} \,|\, \overline{x})) \\
&\to (\nu x)(!x.\mathsf{delay}^t(\mathsf{P} \,|\, \overline{x}) \,|\, \mathsf{delay}^{t-1}(\mathsf{P} \,|\, \overline{x})) \\
&\vdots \\
&\to (\nu x)(!x.\mathsf{delay}^t(\mathsf{P} \,|\, \overline{x}) \,|\, \mathsf{delay}^1(\mathsf{P} \,|\, \overline{x})) \\
&\to \mathsf{P} \,|\, (\nu x)(\overline{x} \,|\, !x.\mathsf{delay}^t(\mathsf{P} \,|\, \overline{x})) \\
&\vdots \\
&\to \mathsf{P} \,|\, \phi^{t+1}(\mathsf{P}) \,|\, (\nu x)(\overline{x} \,|\, !x.\mathsf{delay}^t(\mathsf{P} \,|\, \overline{x})) \\
&\vdots \\
&\to \mathsf{P} \,|\, \phi^{t+1}(\mathsf{P}) \,|\, \phi^{2(t+1)}(\mathsf{P}) \,|\, (\nu x)(\overline{x} \,|\, !x.\mathsf{delay}^t(\mathsf{P} \,|\, \overline{x})) \\
&\vdots
\end{aligned}$$

Here $\phi^n(\mathsf{P})$ abbreviates the n-fold application of $\phi(\cdot)$. If P is timer-free, then $\mathsf{P} \,|\, \phi^{t+1}(\mathsf{P}) \,|\, \phi^{2(t+1)}(\mathsf{P})$ is syntactically identical to $\mathsf{P} \,|\, \mathsf{P} \,|\, \mathsf{P}$. Of course the processes $\phi^n(\mathsf{P})$ can start to interact instead of just waiting for time to pass. But that does not affect the delay operator, because time passes uniformly, regardless of what caused the time steps.

The next example shows that we only need $\mathsf{timer}^1(x(\vec{v}.\mathsf{P}), \mathsf{Q})$ as timing construct. As all others can be built up by iteration of this basic form. Define

$$\mathsf{T}_1 = \mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \qquad \mathsf{T}_{t+1} = \mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{T}_t).$$

Then $\mathsf{T}_t \overset{rc}{\approx} \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q})$ for all $t > 0$. In lieu of a proof for this equation, we show an example of how T_3 and $\mathsf{timer}^3(x(\vec{v}).\mathsf{P}, \mathsf{Q})$ reduce in the same way.

$$\begin{array}{rcl} \mathsf{T}_3 & = & \mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{T}_2) \\ & \to & \phi(\mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{T}_2)) \\ & = & \mathsf{T}_2 \\ & = & \mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{T}_1) \\ & \xrightarrow{x(\vec{y})} & \mathsf{P}\{\vec{y}/\vec{v}\} \end{array}$$

But $\mathsf{timer}^3(x(\vec{v}).\mathsf{P}, \mathsf{Q})$ can match this step by step.

$$\begin{array}{rcl} \mathsf{timer}^3(x(\vec{v}).\mathsf{P}, \mathsf{Q}) & \to & \phi(\mathsf{timer}^3(x(\vec{v}).\mathsf{P}, \mathsf{Q})) \\ & = & \mathsf{timer}^2(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \\ & \xrightarrow{x(\vec{y})} & \mathsf{P}\{\vec{y}/\vec{v}\}. \end{array}$$

The proof of the equality above essentially uses the idea of step by step simulation.

4.4 Equivalences

We have now assembled the basics of π_t. The next step is to investigate its properties. This is an open-ended endeavour and we can only take a first step here, by looking at some canonical equivalences. As pointed out in §2.3.2 there are many candidates, but the maximum sound π_t-theory, should it exist, would be one of the most important. The rest of this chapter is about characterising the maximum sound π_t-theory in various ways, culminating in its presentation as a labelled bisimilarity for some suitable asynchronous transition system.

4.4.1 The Maximum Sound Theory for π_t

We start by defining sound π_t-theories as a consistent and reduction-closed binary congruence on processes that identifies all insensitive processes cf. §2.3.2 and [61]. We want to show that there is a unique maximum such theory which can be obtained by summing all sound π_t-theories. The key issue is to show that this sum is consistent. The overall architecture of the proofs follows [61] but the details differ significantly due to the existence of timers. The presentation below is close to [61] to facilitate comparison. For this reason we are modifying the definition of "sound theory" (cf. 3) slightly. It will be shown later that both definitions induce the same relation.

REMARK 1 *All results in this chapter rely only on constructions that just use local processes. This will allow the transfer of results to other calculi in later chapters. In*

particular, every result also hold for π_t^{loc}, the calculus obtained from π_t by restriction to local processes.

It should be possible to prove a general metalogical result that guarantees the stability of our results under restrictions to local processes.

Basic Definitions and Facts

DEFINITION 22 A *logic* is a pair $\mathcal{L} = (F, \vdash)$ comprising a set F of *formulae* and an *entailment relation* $\vdash \subseteq \mathcal{P}(F) \times F^2$. In this chapter, F will always be pairs of processes in π_t. Such pairs will be often written $\mathsf{P} = \mathsf{Q}$. A set $\mathcal{T}$ of formulae is a *π_t-theory*, or simply a *theory* and its members are *axioms*. We write $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ whenever $(\mathcal{T}, \mathsf{P} = \mathsf{Q}) \in \vdash$ and call $\mathsf{P} = \mathsf{Q}$ a *theorem* or *consequence* of $\mathcal{T}$ in $\mathcal{L}$. If $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ is not derivable, we write $\mathcal{T} \nvdash \mathsf{P} = \mathsf{Q}$. The set of all consequences of $\mathcal{T}$ in $\mathcal{L}$ is denoted $\mid \mathcal{T} \mid_{\mathcal{L}}$. A logic is *monotonic* if $\mathcal{T} \subseteq \mathcal{T}'$ implies $\mid \mathcal{T} \mid \subseteq \mid \mathcal{T}' \mid$. References to the underlying logic $\mathcal{L}$ will often be omitted.

We say *$\mathcal{T}$ is a subtheory of $\mathcal{T}'$* if $|\mathcal{T}| \subseteq |\mathcal{T}'|$. $\mathcal{T}$ is *consistent* if $|\mathcal{T}|$ does not equate all processes, otherwise it is *inconsistent*. $\mathcal{T}$ is *reduction-closed* if $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \twoheadrightarrow \mathsf{P}'$ implies the existence of a reduction sequence $\mathsf{Q} \twoheadrightarrow \mathsf{Q}'$ such that $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$. $\mathcal{T}$ is *strongly reduction-closed* if $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \to \mathsf{P}'$ implies the existence of a reduction $\mathsf{Q} \to \mathsf{Q}'$ such that $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$.

Given a collection $\{\mathcal{T}_i\}_{i \in I}$ of sets of axioms, we write $\Sigma_{i \in I} \mathcal{T}_i$ for the theory obtained from the union of the axiom sets.

DEFINITION 23 A *π_t-logic* $\mathcal{T}$ is a logic where entailment is inductively defined for arbitrary theories $\mathcal{T}$ using the following rules.

1. $(\mathsf{P}, \mathsf{Q}) \in \mathcal{T} \Rightarrow \mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$.
2. $\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$.
3. $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q} \Rightarrow \mathcal{T} \vdash \mathsf{Q} = \mathsf{P}$.
4. $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}, \mathcal{T} \vdash \mathsf{Q} = \mathsf{R} \Rightarrow \mathcal{T} \vdash \mathsf{P} = \mathsf{R}$.
5. If $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $C[\cdot]$ is a context then $\mathcal{T} \vdash C[\mathsf{P}] = C[\mathsf{Q}]$.

Unlike most of our other definitions towards a notion of reduction congruence, the account of π_t-congruence above is not canonical because it has a non-trivial variant that is equally appealing. Legitimate variation concerns closure under (non-injective) renaming. In Definition 23 we stipulated $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ to imply $\mathcal{T} \vdash C[\mathsf{P}] = C[\mathsf{Q}]$ for arbitrary contexts $C[\cdot]$. This requires not only

$$\mathcal{T} \vdash \mathsf{P} = \mathsf{Q} \Rightarrow \begin{cases} \mathcal{T} \vdash x(\vec{v}).\mathsf{P} = x(\vec{v}).\mathsf{Q} \\ \mathcal{T} \vdash !x(\vec{v}).\mathsf{P} = !x(\vec{v}).\mathsf{Q} \\ \mathcal{T} \vdash \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{R}) = \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R}) \end{cases}$$

to hold but also implies (Proposition 4)

$$\mathcal{T} \vdash \mathsf{P} = \mathsf{Q} \Rightarrow \mathcal{T} \vdash \mathsf{P}\{\vec{x}/\vec{v}\} = \mathsf{Q}\{\vec{x}/\vec{v}\}.$$

We could follow [61] and be less demanding. Instead of requiring π_t to be closed under

$$\mathcal{T} \vdash \mathsf{P} = \mathsf{Q} \Rightarrow \begin{cases} \mathcal{T} \vdash x(\vec{v}).\mathsf{P} = x(\vec{v}).\mathsf{Q} \\ \mathcal{T} \vdash !x(\vec{v}).\mathsf{P} = !x(\vec{v}).\mathsf{Q} \\ \mathcal{T} \vdash \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{R}) = \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R}) \end{cases}$$

we would simply ask the following to hold.

$$(\forall\{\vec{x}/\vec{v}\}.\mathcal{T} \vdash \mathsf{P}\{\vec{x}/\vec{v}\} = \mathsf{Q}\{\vec{x}/\vec{v}\}) \Rightarrow \begin{cases} \mathcal{T} \vdash x(\vec{v}).\mathsf{P} = x(\vec{v}).\mathsf{Q} \\ \mathcal{T} \vdash !x(\vec{v}).\mathsf{P} = !x(\vec{v}).\mathsf{Q} \\ \mathcal{T} \vdash \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{R}) = \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R}) \end{cases}$$

For various π-calculi this alternative account of congruence yields different equivalence. For asynchronous π-calculi, our Definition 23 is somewhat more convenient, because it allows a nice characterisation of the largest sound theory as barbed congruence. (Theorem 21), while the "less demanding" definition "scales" more gracefully to synchronous calculi or the inclusion of unrestricted sums. The question of the coincidence of the two induced equivalences in π_t-calculus is open.

LEMMA 9 *1. A binary relation $\mathcal{R}$ on processes is a* π_t-congruence *iff* $\mathcal{R} = |\, \mathcal{R} \,|$.

2. Each π_t-logic is monotonic.

3. $|\, \mathcal{T} \,| \vdash \mathsf{P} = \mathsf{Q}$ *if and only if* $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$*; i.e.* $||\, \mathcal{T} \,|| = |\, \mathcal{T} \,|$.

4. Assume $\mathcal{T}$ is a consistent, reduction-closed congruence, containing $\equiv$. Then $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ *iff* $(\mathsf{P}, \mathsf{Q}) \in \mathcal{T}$.

PROOF: By induction on the derivation of consequences. □

Note that in (3) above, $||\cdot||$ means applying $|\cdot|$ twice. The next lemma provides some useful characterisations of reduction-closure and also paves the way for a better understanding of an important operation on theories, their sum.

LEMMA 10 *Let $\mathcal{T}$ be a π_t theory.*

1. *$\mathcal{T}$ is reduction-closed if and only if, whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$, then, for all contexts $C[\cdot]$, $C[\mathsf{P}] \twoheadrightarrow \mathsf{P}'$ implies $C[\mathsf{Q}] \twoheadrightarrow \mathsf{Q}'$, for some Q' with $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$.*

2. *$\mathcal{T}$ is reduction-closed if and only if, whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$, then, for all contexts $C[\cdot]$, $C[\mathsf{P}] \rightarrow \mathsf{P}'$ implies $C[\mathsf{Q}] \twoheadrightarrow \mathsf{Q}'$, for some Q' with $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$.*

3. *Let* $\mathcal{T} = \Sigma_{i\in I}\mathcal{T}_i$. $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ *if and only if there are* $i_1, ..., i_n \in I$ *such that*

$$\mathcal{T}_{i_1} \vdash \mathsf{P}_0 = \mathsf{P}_1,\ ...\mathcal{T}_{i_{n-1}} \vdash \mathsf{P}_{n-2} = \mathsf{P}_{n-1},\ \mathcal{T}_{i_n} \vdash \mathsf{P}_{n-1} = \mathsf{P}_n,$$

where $\mathsf{P} = \mathsf{P}_0$ *and* $\mathsf{P}_n = \mathsf{Q}$.

4. *If* $\mathcal{T}_i$ *is reduction-closed for all* $i \in I$, *then so is* $\Sigma_{i\in I}\mathcal{T}_i$.

5. $|\,\mathcal{T}_i\,| \subseteq |\Sigma_{i\in I}\,\mathcal{T}_i\,|$ *for all* i.

PROOF: (1) and (2) can be proven exactly as [61, Proposition 3.2] by nested inductions on the structure of $C[\cdot]$, on the number of reduction steps in $C[\mathsf{P}] \twoheadrightarrow \mathsf{P}'$ and on the derivation of these reduction steps. The proof of (3) follows [61, Lemma 3.3]: one direction is by induction on the derivation of $\Sigma_{i\in I}\mathcal{T}_i \vdash \mathsf{P} = \mathsf{Q}$ while the reverse follows from the monotonicity of π_t-logics. In all cases, the adaptation to the extended syntax is straightforward. (4) is an immediate consequence of (3) while (5) is immediate, using the monotonicity of π_t-logics again. □

Insensitivity and Sound Theories

As we show later (Theorem 18), there is no maximum consistent, reduction-closed theory. However, if we are just a tiny bit more demanding and also ask our reduction-closed theories to identify at least a substantial number of processes that could not possibly interact with their environment, then the problem disappears. The beauty of this solution, which was adapted from λ-calculi [11] to processes by Honda and Yoshida [61], is that it does not require any form of observation predicate (unless you count $\mathsf{an}(\cdot)$). It can be defined solely with reference to equations that we expect to be contained in any reasonable reference anyway.

DEFINITION 24 A process P is *insensitive* if $\mathsf{P} \twoheadrightarrow \mathsf{Q}$ implies $\mathsf{an}(\mathsf{Q}) = \emptyset$.

The next lemma suggests an alternative definition of insensitivity that does not require reference to free names. We conjecture that it induces the same equivalence as our definition.

LEMMA 11 *If* P *is insensitive then the following holds. For all reduction contexts* $C[\cdot]$, *if* $C[\mathsf{P}] \twoheadrightarrow \mathsf{Q}$ *then there is a reduction context* $C'[\cdot]$ *and an insensitive process* P' *such that* $\mathsf{P} \twoheadrightarrow \mathsf{P}'$, $C[\cdot] \twoheadrightarrow C'[\cdot]$ *and* $\mathsf{Q} \equiv C[\mathsf{P}']$. *In particular, if* $\mathsf{P} \twoheadrightarrow \mathsf{Q}$ *and* P *is insensitive, then so is* Q. *The converse does not hold.*

PROOF: By inductions on the number of reduction steps in $C[\mathsf{P}] \twoheadrightarrow \mathsf{Q}$, the structure of $C[\cdot]$ and the derivation of individual reduction steps. For a counterexample, consider $\mathsf{P} = \mathsf{fw}_{xx}$. □

DEFINITION 25 A π_t theory is *sound* if it is consistent, reduction-closed and equates all insensitive terms.

The following lemma summarises some elementary facts about sound theories that will be useful later.

LEMMA 12 *Let $\mathcal{T}$ be a sound theory. Assume $\mathcal{T} \vdash \mathsf{P}_1 = \mathsf{P}_2$, $\mathcal{T} \vdash \mathsf{Q}_1 = \mathsf{Q}_2$, $\mathsf{P}_i \bowtie \mathsf{Q}_i$ for $i = 1, 2$ and $\mathsf{P}_1|\mathsf{Q}_1 \twoheadrightarrow_{n_1} \mathsf{R}_1$. By definition, this implies the existence of a reduction sequence $\mathsf{P}_2|\mathsf{Q}_2 \twoheadrightarrow_{n_2} \mathsf{R}_2$ such that $\mathcal{T} \vdash \mathsf{R}_1 = \mathsf{R}_2$. In addition*

1. *$R_i \equiv \mathsf{P}'_i|\mathsf{Q}'_i$, $\mathsf{P}'_i \bowtie \mathsf{Q}'_i$, $\mathsf{P}_i \twoheadrightarrow_{n_i} \mathsf{P}'_i$ and $\mathsf{Q}_i \twoheadrightarrow_{n_i} \mathsf{Q}'_i$ for $i = 1, 2$.*
2. *If $\mathsf{P}_1 \bowtie \mathsf{Q}_2$ and $\mathsf{P}_2 \bowtie \mathsf{Q}_1$, then $\mathcal{T} \vdash \mathsf{P}'_1 = \mathsf{P}'_2$ and $\mathcal{T} \vdash \mathsf{Q}'_1 = \mathsf{Q}'_2$.*

PROOF: (1) is immediate by reduction-closure and Lemma 8.3. For (2), let $\{\vec{x}\} = \mathsf{fn}(\mathsf{P}'_1) \cup \mathsf{fn}(\mathsf{P}'_2)$. By assumptions, we can derive

$$\begin{aligned} \mathcal{T} \vdash \mathsf{P}'_1|\mathsf{Q}'_1 = \mathsf{P}'_2|\mathsf{Q}'_2 &\Rightarrow \mathcal{T} \vdash (\nu\vec{x})(\mathsf{P}'_1|\mathsf{Q}'_1) = (\nu\vec{x})(\mathsf{P}'_2|\mathsf{Q}'_2) \\ &\Rightarrow \mathcal{T} \vdash \mathsf{Q}'_1|(\nu\vec{x})\mathsf{P}'_1 = \mathsf{Q}'_2|(\nu\vec{x})\mathsf{P}'_2 \\ &\Rightarrow \mathcal{T} \vdash \mathsf{Q}'_1|\mathsf{0} = \mathsf{Q}'_2|\mathsf{0} \\ &\Rightarrow \mathcal{T} \vdash \mathsf{Q}'_1 = \mathsf{Q}'_2 \end{aligned}$$

as $(\nu\vec{x})\mathsf{P}'_i$ is insensitive. $\mathcal{T} \vdash \mathsf{P}'_1 = \mathsf{P}'_2$ can be established mutatis mutandis. □

Of course we would like to know just what kind of processes are equated by sound theories. Unfortunately, this question does not have a straightforward answer. Instead we begin at the "opposite end" and exhibit some processes that will never be equated.

DEFINITION 26 Processes P and Q are *incompatible*, written $\mathsf{P} \# \mathsf{Q}$ if for all sound theories $\mathcal{T}$: $\mathcal{T} \not\vdash \mathsf{P} = \mathsf{Q}$.

LEMMA 13 1. *If $\mathsf{P}\Downarrow_x$ but $\mathsf{Q}\not\Downarrow_x$ then $\mathsf{P} \# \mathsf{Q}$.*

2. *Let $\mathcal{T}$ be sound. If $\mathcal{T} \vdash P = Q$, then $\mathsf{P}\Downarrow_x$ if and only if $\mathsf{Q}\Downarrow_x$.*
3. *If $C[\cdot]$ is a context and $C[\mathsf{P}] \# C[\mathsf{Q}]$ then $\mathsf{P} \# \mathsf{Q}$.*

PROOF: For (1), choose a sound theory $\mathcal{T}$ and assume R is an arbitrary process with $\mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q}) = \{\vec{y}\}$. Let z be a fresh name and define

$$C[\cdot] = (\nu z)(z.\mathsf{R} \,|\, (\nu\vec{y})([\cdot] \,|\, x(\vec{v}).\overline{z})).$$

Clearly, $C[\mathsf{P}] \twoheadrightarrow \mathsf{R} \,|\, (\nu\vec{y})\mathsf{P}'$ for some appropriate P'. Clearly, $(\nu\vec{y})\mathsf{P}'$ is insensitive, so $\mathcal{T} \vdash \mathsf{R} \,|\, (\nu\vec{y})\mathsf{P}' = \mathsf{R}$. On the other hand $C[\mathsf{Q}]$ is insensitive, as one can easily show, so by reduction-closure and soundness, $\mathcal{T} \vdash \mathsf{R} = \mathsf{0}$ for all R, in contradiction to the consistency of $\mathcal{T}$. Now (2) is an immediate consequence of (1) and (3) is easy because $\mathcal{T} \vdash P = Q$ is by definition closed under contexts. □

Making Use of Time: Transition- and Reduction Spotters

We have now assembled enough tools to embark on contrasting sound theories with those that are merely consistent and reduction-closed. We begin by sharpening Lemma 13.

PROPOSITION 2 *1. If* $\mathsf{P}\downarrow_x$ *but* $\mathsf{Q}\not\downarrow_x$ *then* $\mathsf{P}\,\#\,\mathsf{Q}$.

2. Let $\mathcal{T}$ *be sound. If* $\mathcal{T}\vdash P=Q$, *then* $\mathsf{P}\downarrow_x$ *if and only if* $\mathsf{Q}\downarrow_x$.

PROOF: Assume $\mathcal{T}$ is a sound theory. Choose an arbitrary process R and let a be a fresh name. Define

$$C[\cdot]=(\nu a)(a.\mathsf{R}\,|\,(\nu\vec{y})([\cdot]\,|\,\mathsf{timer}^1(x(\vec{v}).\overline{a},0)))$$

where $\{\vec{y}\}=\mathsf{fn}(\mathsf{P})\cup\mathsf{fn}(\mathsf{Q})$. Then

$$C[\mathsf{P}]\rightarrow(\nu a)(a.\mathsf{R}\,|\,\overline{a}\,|\,(\nu\vec{y})\mathsf{P}')\rightarrow\mathsf{R}\,|\,(\nu\vec{y})\mathsf{P}'\qquad\text{where}\qquad\mathcal{T}\vdash\mathsf{R}=\mathsf{R}\,|\,(\nu\vec{y})\mathsf{P}'$$

for some appropriate P' such that $(\nu\vec{y})\mathsf{P}'$ is insensitive. On the other hand, $C[\mathsf{Q}]$ has only one form of reduction step:

$$C[\mathsf{Q}]\rightarrow(\nu a)(a.\mathsf{R}\,|\,(\nu\vec{y})\mathsf{Q}')\qquad\text{because}\qquad\mathsf{Q}\rightarrow\mathsf{Q}'.$$

Hence $C[\mathsf{Q}]$ is insensitive, so reduction-closure implies $\mathcal{T}\vdash\mathsf{R}=\mathsf{0}$ for all R, in violation of $\mathcal{T}$'s consistency. This establishes (1) with (2) being an immediate consequence. □

The kind of reasoning used in the proof of Proposition 2 will be pervasive in the rest of this chapter. To make life easier we summarise the main tools in the following definition.

DEFINITION 27 Let $t>0$ and x,y,z be names such that $a\notin\{x,y,z\}$.

$$\mathsf{RS}^t_{xy}=(\nu a)(\overline{a}\,|\,\mathsf{timer}^t(a.\overline{x},\overline{y}))\qquad\mathsf{RS}^t_{xyz}=(\nu a)(\overline{a}\,|\,\mathsf{timer}^t(a.\overline{x},\mathsf{RS}^1_{yz}))$$

These processes are called *reduction spotters.*

The content of the next lemma is trivial but almost all subsequent proofs in this chapter rely on it or could be made to do so. In fact, it is possible to base an axiomatic treatment of timed process calculi almost exclusively on the properties of RS^1_{xyz}, as listed below, although we will not do so in this text.

LEMMA 14 *Let* x,y,z *be distinct names. If* $\mathsf{RS}^t_{xyz}\twoheadrightarrow_n\mathsf{P}$ *then exactly one of the following is true.*

(1) $\mathsf{P}\equiv\overline{x},n\geq 1$. (2) $\mathsf{P}\equiv\overline{y},n>t$. (3) $\mathsf{P}\equiv\overline{z},n>t$.
(4) $\mathsf{P}\equiv\mathsf{RS}^1_{yz},n=t$. (5) $\mathsf{P}\equiv\mathsf{RS}^{t-n}_{xyz},0\leq n<t$.

In addition, the following implications hold.

(6) $\mathsf{P}\downarrow_x \Rightarrow 0 < n \leq t.$
(7) $\mathsf{P}\downarrow_y \Rightarrow n = t+1.$
(8) $\mathsf{P}\downarrow_z \Rightarrow n = t+1.$
(9) $\mathsf{P}\Downarrow_x \Rightarrow 0 \leq n \leq t.$
(10) $(\mathsf{P}\Downarrow_y \textit{ or } \mathsf{P}\Downarrow_z) \Rightarrow 0 \leq n \leq t+1.$
(11) $(\mathsf{P}\Downarrow_{xy} \textit{ or } \mathsf{P}\Downarrow_{xz} \textit{ or } \mathsf{P}\Downarrow_{yz}) \Rightarrow 0 \leq n \leq t.$

PROOF: Immediate from the definitions. □

The next lemma shows how drastically the addition of timers changes the resulting theory: reduction-closure collapses into strong reduction-closure.

PROPOSITION 3 *Let $\mathcal{T}$ be a consistent theory that identifies all insensitive terms. Then the following statements are equivalent.*

1. *$\mathcal{T}$ is reduction-closed.*
2. *$\mathcal{T}$ is strongly reduction-closed.*
3. *Whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \twoheadrightarrow_n \mathsf{P}'$ then there must be Q' such that $\mathsf{Q} \twoheadrightarrow_n \mathsf{Q}'$ and $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$.*
4. *For all contexts $C[\cdot]$: whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $C[\mathsf{P}] \rightarrow \mathsf{P}'$ then some Q' must exist such that $C[\mathsf{Q}] \rightarrow \mathsf{Q}'$ and $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$.*

PROOF: We only show the non-trivial implication ($1 \Rightarrow 2$). From the assumptions, $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \rightarrow \mathsf{P}'$ we can infer

$$\mathcal{T} \vdash \mathsf{P}|\mathsf{RS}^1_{xyz} = \mathsf{Q}|\mathsf{RS}^1_{xyz} \qquad \text{and} \qquad \mathsf{P}|\mathsf{RS}^1_{xyz} \rightarrow \mathsf{P}'|\mathsf{RS}^1_{yz}$$

where x, y, z are distinct and fresh names. Then reduction-closure together with Lemma 12 guarantees a reduction sequence

$$\mathsf{Q}|\mathsf{RS}^1_{xyz} \twoheadrightarrow_n \mathsf{Q}'|\mathsf{R}'$$

where $\mathsf{Q} \twoheadrightarrow_n \mathsf{Q}'$, $\mathsf{RS}^1_{xyz} \twoheadrightarrow_n \mathsf{R}'$, $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$ and $\mathcal{T} \vdash \mathsf{RS}^1_{yz} = \mathsf{R}'$. Then $\mathsf{R}' \Downarrow_{yz}$, so $\mathsf{R}' \equiv \mathsf{RS}^1_{yz}$ and $n = 1$ (by Lemmas 13 and 14). □

Next we establish a crucial property of sound theories that follows from congruency, even in untimed calculi. However, due to the possibility of idling, we need to engage timers in the proof.

PROPOSITION 4 *(renaming-closure) If $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ then for all appropriate $\vec{x}$, $\vec{v}$: $\mathcal{T} \vdash \mathsf{P}\{\vec{x}/\vec{v}\} = \mathsf{Q}\{\vec{x}/\vec{v}\}$.*

PROOF: From the assumptions we can infer

$$\mathcal{T} \vdash (\nu a)(\overline{a}\langle\vec{x}\rangle \,|\, \mathsf{timer}^1(a(\vec{v}).\mathsf{P}, \overline{b})) = (\nu a)(\overline{a}\langle\vec{x}\rangle \,|\, \mathsf{timer}^1(a(\vec{v}).\mathsf{Q}, \overline{b}))$$

where a, b are fresh and distinct. The process on the left of this equation reduces to $\mathsf{P}\{\vec{x}/\vec{v}\}$ in one step. In the light of Proposition 2 and strong reduction-closure, the only way this reduction can be matched is if the process on the right one-step reduces to $\mathsf{Q}\{\vec{x}/\vec{v}\}$, in which case $\mathcal{T} \vdash \mathsf{P}\{\vec{x}/\vec{v}\} = \mathsf{Q}\{\vec{x}/\vec{v}\}$, as required. □

We can now also strengthen Lemma 12.

LEMMA 15 *Let $\mathcal{T}$ be a sound theory. Assume $\mathcal{T} \vdash \mathsf{P}_1 = \mathsf{P}_2$, $\mathcal{T} \vdash \mathsf{Q}_1 = \mathsf{Q}_2$, $\mathsf{P}_i \bowtie \mathsf{Q}_i$ for $i = 1, 2$ and $\mathsf{P}_1|\mathsf{Q}_1 \twoheadrightarrow_n \mathsf{R}_1$. This implies the existence of a reduction sequence $\mathsf{P}_2|\mathsf{Q}_2 \twoheadrightarrow_n \mathsf{R}_2$ such that $\mathcal{T} \vdash \mathsf{R}_1 = \mathsf{R}_2$. In addition*

1. *$R_i \equiv \mathsf{P}'_i|\mathsf{Q}'_i$, $\mathsf{P}'_i \bowtie \mathsf{Q}'_i$, $\mathsf{P}_i \twoheadrightarrow_n \mathsf{P}'_i$ and $\mathsf{Q}_i \twoheadrightarrow_n \mathsf{Q}'_i$ for $i = 1, 2$.*
2. *If in addition $\mathsf{P}_1 \bowtie \mathsf{Q}_2$ and $\mathsf{P}_2 \bowtie \mathsf{Q}_1$, then $\mathcal{T} \vdash \mathsf{P}'_1 = \mathsf{P}'_2$ and $\mathcal{T} \vdash \mathsf{Q}'_1 = \mathsf{Q}'_2$.*

PROOF: By induction on n. For the inductive step assume $\mathsf{P}_1|\mathsf{Q}_1 \to \mathsf{R}_1 \twoheadrightarrow_n \mathsf{R}'_1$. By Proposition 3 there is R_2 with $\mathcal{T} \vdash \mathsf{R}_1 = \mathsf{R}_2$ where $\mathsf{P}_2|\mathsf{Q}_2 \to \mathsf{R}_2$. By Lemma 8.3, $\mathsf{R}_i \equiv \mathsf{P}'_i|\mathsf{Q}'_i$, $\mathsf{P}_i \to \mathsf{P}'_i$, $\mathsf{Q}_i \to \mathsf{Q}'_i$ and $\mathsf{P}'_i \bowtie \mathsf{Q}'_i$. Now apply the (IH). The proof of (2) can be lifted verbatim from the corresponding proof of Lemma 12.2. □

The following proposition proves that sound theories behave gracefully under time-passing, in the sense that equations are not invalidated by the application of ϕ.

PROPOSITION 5 *If $\mathcal{T}$ is a sound theory, then $\mathcal{T}$ is time-closed.*

PROOF: Assume that $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$. Define

$$C[\cdot] = [\cdot] \,|\, (\nu x)(x.\overline{a} \,|\, \overline{x})$$

where a is a fresh name. Then $C[\mathsf{P}] \to \phi(\mathsf{P}) \,|\, \overline{a}$ and by strong reduction-closure $C[\mathsf{Q}] \to \mathsf{Q}'$ for some appropriate Q' with $\mathsf{Q}' \downarrow_a$ (Proposition 2.2). It is easy to see that this is only possible if $\mathsf{Q}' \equiv \phi(\mathsf{Q}) \,|\, \overline{a}$. Applying Lemma 15 then yields $\mathcal{T} \vdash \phi(\mathsf{P}) = \phi(\mathsf{Q})$. □

Existence of the Maximum Sound Theory

As pointed out before, the key issue in proving that the maximum sound theory exists is to show that soundness is preserved under the formation of sums. The next definition and lemma do just that.

DEFINITION 28 Let S be a set of processes. A theory $\mathcal{T}$ *isolates* S if $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \in S$ together imply $\mathsf{Q} \in S$.

LEMMA 16 *Let $\mathcal{T}$ be a theory.*

1. *If $\mathcal{T}$ isolates a set that is neither empty nor universal, then $\mathcal{T}$ is consistent.*
2. *If $\mathcal{T}_i$ isolates S for each $i \in I$ then $\Sigma_{i\in I}\mathcal{T}_i$ also isolates S.*
3. *Let $\mathcal{T}$ be sound. Then $\mathcal{T}$ isolates the sets $\{\mathsf{P} \mid \mathsf{P} \Downarrow_x\}$ and $\{\mathsf{P} \mid \mathsf{P} \downarrow_x\}$ for each name x, neither of which is empty or contains all processes.*
4. *Let $\mathcal{T}_i$ be sound for each $i \in I$. Then $\Sigma_{i\in I}\mathcal{T}_i$ is also sound.*

PROOF: (1) is immediate from the definitions. (2) is immediate by Lemma 10.3, (3) is by Lemma 13.2. In the light of 10.4, we need only show that $\Sigma_{i\in I}\mathcal{T}_i$ is consistent to establish (4): choose $x \in \mathcal{N}$. By (3) we know that $\mathcal{T}_i$ isolates $\{\mathsf{P} \mid \mathsf{P}\Downarrow_x\}$, so $\Sigma_{i\in I}\mathcal{T}_i$ does the same by (2). Now (1) guarantees soundness. □

We can now state our first major result in this chapter and prove half of it.

THEOREM 18
1. *Define $\mathcal{T}_{\max} = \Sigma\{\mathcal{T} \mid \mathcal{T}$ is a sound theory$\}$. Then $\mathcal{T}_{\max}$ is the unique sound theory such that $\mathcal{T} \subseteq \mathcal{T}_{\max}$ for all sound theories $\mathcal{T}$. $\mathcal{T}_{\max}$ is called the* maximum sound theory*.*
2. *There is no maximum consistent, reduction-closed theory.*

PROOF: By construction, $\mathcal{T}_{max}$ contains every sound theory as a subset, so we need only establish soundness. But that follows from Lemma 16.4. (2) will be shown in Section 4.5. □

Decomposition Theorems

Decomposition Theorems are sometimes convenient for reasoning. They allow to conclude that processes P and Q are equal, provided $C[\mathsf{P}]$ and $C[\mathsf{Q}]$ are. Of course $C[\cdot]$ cannot range over all context, just over some restricted class, for otherwise the equality holding between $!x|!x$ and $x|!x$ in all reasonable equivalences would force $!x$ and x to be equated, too. But that would not be good! Another problematic example is that we have $\mathcal{T} \vdash \overline{x}\langle\vec{y}\rangle \,|\, \overline{x}\langle\vec{y}\rangle = \mathsf{delay}^1(\overline{x}\langle\vec{y}\rangle) \,|\, \overline{x}\langle\vec{y}\rangle$ but not $\mathcal{T} \vdash \overline{x}\langle\vec{y}\rangle = \mathsf{delay}^1(\overline{x}\langle\vec{y}\rangle)$ as we shall verify in Example 7 later.

The main problem with Decomposition Theorems is that they are hard to come by. At the time of writing, they are only known in very restricted forms, such as Lemmas 12 and 15. Although we believe that a better understanding of just what kind of decompositions are and are not possible would be desirable, we do not to lift that stone here. Instead we prove some rather peculiar special cases that will be vital later on.

How are Decomposition Theorems proven? It seems unreasonable to expect that $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ just because $\mathcal{T} \vdash C[\mathsf{P}] = C[\mathsf{Q}]$ – sound theories are not required

to be closed under decomposition. We will have to construct an appropriate second theory from the first for this purpose. The next lemma gives some tools for showing how to verify that the candidate relations are indeed sound theories.

DEFINITION 29 Let $\mathcal{R}$ be a binary relation on processes. Then

$$C[\mathcal{R}] \quad = \quad \{(C[\mathsf{P}], C[\mathsf{Q}]) \,|\, (\mathsf{P}, \mathsf{Q}) \in \mathcal{R}, C[\cdot] \text{ is an arbitrary context}\}$$

In addition, please recall $(\cdot)^*$, the transitive closure operation.

$\mathcal{R}$ *preserves context-closure* if $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ implies $(C[\mathsf{P}], C[\mathsf{Q}]) \in \mathcal{R}$ for all $C[\cdot]$. $\mathcal{R}$ has *strong-barb-preservation* if $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ implies: $\mathsf{P}\downarrow_x \Leftrightarrow \mathsf{Q}\downarrow_x$.

A function F on binary relations of processes is *expansive* if $\mathcal{R} \subseteq F(\mathcal{R})$ for all appropriate $\mathcal{R}$. F *preserves* $\#$ if $\mathcal{R} \cap \# = \emptyset$ implies $F(\mathcal{R}) \cap \# = \emptyset$. F *preserves simple inequalities* if $(\overline{x}\langle\vec{y}\rangle, \mathsf{0}) \notin \mathcal{T}$ implies $(\overline{x}\langle\vec{y}\rangle, \mathsf{0}) \notin F(\mathcal{T})$. F *preserves context-closure* if the context-closure of $F(\mathcal{R})$ is implied by that of $\mathcal{R}$. F *preserves time-closure* if $\mathcal{R}$'s being time-closed, implies time-closure of $F(\mathcal{R})$. F *preserves strong reduction-closure* if $\mathcal{R}$'s strong reduction-closure implies that of $F(\mathcal{R})$. F *preserves strong-barb-preservation*, if the strong-barb-preservation of $\mathcal{R}$ implies that of $F(\mathcal{R})$.

LEMMA 17 *The functions $C[\cdot]$ and $(\cdot)^*$ have the following properties.*

- *$C[\cdot]$ is expansive and preserves symmetry, time-closure, $\#$, strong-barb-preservation, simple inequalities and context-closure. In addition, if $\mathcal{R}$ is strongly reduction-closed, time-closed, contains $\equiv$ and $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ implies $(\mathsf{P}|\mathsf{R}, \mathsf{Q}|\mathsf{R}) \in \mathcal{R}$ for all R (we call this property* |-closure*), then $C[\mathcal{R}]$ is strongly reduction-closed.*

- *$(\cdot)^*$ is expansive and preserves symmetry, transitivity, time-closure, context-closure, $\#$, strong-barb-preservation and strong reduction-closure. In addition, when restricted to strong barb preserving relations, $(\cdot)^*$ preserves simple inequalities.*

PROOF: It is immediate that $C[\cdot]$ and $(\cdot)^*$ are expansive and preserve symmetry.

For $C[\cdot]$'s preservation of simple inequalities, assume that $(\overline{x}\langle\vec{y}\rangle, \mathsf{0}) \notin \mathcal{R}$. If we had $(\overline{x}\langle\vec{y}\rangle, \mathsf{0}) \in C[\mathcal{R}]$, that mean there was a context $C'[\cdot]$ and $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ such that $C'[\mathsf{P}] = \mathsf{0}$ and $C'[\mathsf{Q}] = \overline{x}\langle\vec{y}\rangle$. A quick induction on the structure of $C'[\cdot]$ show that this is impossible without $(\overline{x}\langle\vec{y}\rangle, \mathsf{0}) \in \mathcal{R}$, contradicting our assumption. Next, assume that $\mathsf{P}\#\mathsf{Q}$, $(\mathsf{P}, \mathsf{Q}) \notin \mathcal{R}$ but $(\mathsf{P}, \mathsf{Q}) \in C[\mathcal{R}]$. This implies the existence of $(\mathsf{P}', \mathsf{Q}') \in \mathcal{R}$ and a context $C[\cdot]$ such that $\mathsf{P} = C[\mathsf{P}']$ and $\mathsf{Q} = C[\mathsf{Q}']$. If $\# \cap \mathcal{R} = \emptyset$ then $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$ for some sound theory $\mathcal{T}$, hence also $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$, violating our assumptions. But then $C[\cdot]$ must preserve $\#$. Finally, assume that $(\mathsf{P}_0, \mathsf{P}_n) \in \mathcal{R}^* \cap \#$ while $\# \cap \mathcal{R} = \emptyset$. Then we can find $(\mathsf{P}_0, \mathsf{P}_1), ..., (\mathsf{P}_{n-1}, \mathsf{P}_n) \in \mathcal{R}$ with $\mathsf{P}_i \# \mathsf{P}_{i+1}$. Hence there must be sound theories $(\mathcal{T}_i)$ such that $\mathcal{T}_i \vdash \mathsf{P}_i = \mathsf{P}_{i+1}$. But $\Sigma_i \mathcal{T}_i$ is a sound theory (Lemma 16.4) with $\Sigma\mathcal{T}_i \vdash \mathsf{P}_0 = \mathsf{P}_n$, which is a contradiction. This means that

$(\cdot)^*$, too, preserves $\#$. To see that $(\cdot)^*$ preserves simple inequalities, let $(\mathsf{0}, \overline{x}\langle\vec{y}\rangle) \notin \mathcal{R}$ but $(\mathsf{0}, \overline{x}\langle\vec{y}\rangle) \in \mathcal{R}^*$, in other words there must be $(\mathsf{P}_i, \mathsf{Q}_i) \in \mathcal{R}, (i = 1, ..., n)$ such that $\mathsf{0} = \mathsf{P}_1$, $\mathsf{Q}_i = \mathsf{P}_{i+1}$ and $\mathsf{Q}_n = \overline{x}\langle\vec{y}\rangle$. But then we can find $i \in \{1, ..., n\}$ such that $\mathsf{P}_i \not\downarrow_x$ but $\mathsf{Q}_i \downarrow_x$, violating $\mathcal{R}$'s preservation of strong barbs.

Regarding $C[\cdot]$'s preservation of strong-barb-preservation, assume that $(C[\mathsf{P}], C[\mathsf{Q}]) \in C[\mathcal{R}]$. If $C[\cdot]$ is not a reduction context, then clearly $C[\mathsf{P}]\downarrow_x \Leftrightarrow C[\mathsf{Q}]\downarrow_x$. Otherwise, if $C[\cdot]\downarrow_x$, then $C[\mathsf{P}]\downarrow_x$ and $C[\mathsf{Q}]\downarrow_x$. Finally, if $C[\cdot]\not\downarrow_x$, then $C[\mathsf{P}]\downarrow_x \Leftrightarrow \mathsf{P}\downarrow_x \Leftrightarrow \mathsf{Q}\downarrow_x \Leftrightarrow C[\mathsf{Q}]\downarrow_x$, as a straightforward induction on the derivation of barbs shows. For $(\cdot)^*$, if we have $(\mathsf{P}_0, \mathsf{P}_n) \in \mathcal{R}^*$ because $(\mathsf{P}_0, \mathsf{P}_1), ..., (\mathsf{P}_{n-1}, \mathsf{P}_n) \in \mathcal{R}$, and $\mathsf{P}_0\downarrow_x$ then $\mathcal{R}$'s strong-barb-preservation allows to infer $\mathsf{P}_1\downarrow_x, ..., \mathsf{P}_n\downarrow_x$.

Time-closure of $C[\cdot]$ comes from Lemma 6, while that of $(\cdot)^*$ is immediate from the definitions.

Preservation of transitivity by $(\cdot)^*$ is immediate.

For context-closure of $(\cdot)^*$, choose a context $C[\cdot]$ and let $(\mathsf{P}_0, \mathsf{P}_{n+1}) \in C[\mathcal{R}]$ because $(\mathsf{P}_0, \mathsf{P}_1), ..., (\mathsf{P}_n, \mathsf{P}_{n+1}) \in \mathcal{R}$. Then $(C[\mathsf{P}_0], C[\mathsf{P}_1]), ..., (C[\mathsf{P}_n], C[\mathsf{P}_{n+1}]) \in \mathcal{R}$ by assumption, so $(C[\mathsf{P}_0], C[\mathsf{P}_{n+1}]) \in \mathcal{R}^*$. Context-closure of $C[\cdot]$ is by definition.

Assume $\mathcal{R}$ is strongly reduction-closed and $\mathsf{P}\rightarrow\mathsf{P}'$ in the remainder of this paragraph. If $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}^*$ there there must be $(\mathsf{P}_0, \mathsf{P}_1), ..., (\mathsf{P}_{n-1}, \mathsf{P}_n) \in \mathcal{R}$ where $\mathsf{P}_0 = \mathsf{P}$ and $\mathsf{P}_n = \mathsf{Q}$. Iterating $\mathcal{R}$'s reduction closure, we get reductions $\mathsf{P}_0\rightarrow\mathsf{P}'_0, ..., \mathsf{P}_{n-1}\rightarrow\mathsf{P}'_n$ with $(\mathsf{P}'_0, \mathsf{P}'_1), ..., (\mathsf{P}'_{n-1}, \mathsf{P}'_n) \in \mathcal{R}$. Thus $(\mathsf{P}'_0, \mathsf{P}'_n) \in (\mathcal{R})^*$, as required, showing the preservation of strong reduction-closure by $(\cdot)^*$. Finally, we deal with $C[\cdot]$ and its relation to strong reduction-closure. Let $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ and assume that $C[\mathsf{P}] \rightarrow \mathsf{P}'$. Here $C[\cdot]$ is an arbitrary context, not to be confused with, or dependent on the function $C[\cdot]$. We proceed by induction on the structure of $C[\cdot]$. We repeatedly use Lemma 6 tacitly.

$\underline{C[\cdot] = [\cdot]}$. Immediate by strong reduction closure of $\mathcal{R}$.

$\underline{C[\cdot] = C'[\cdot]|\mathsf{R}}$. We use induction on the derivation on $C[\mathsf{P}] \rightarrow \mathsf{P}'$.

- $\underline{\mathsf{R} \rightarrow \mathsf{R}' \text{ implies } C'[\mathsf{P}]|\mathsf{R} \rightarrow \phi(C'[\mathsf{P}])|\mathsf{R}'}$. We have two cases. If $C[\cdot]$ is a reduction-context, then reduction is $C'[\mathsf{P}]|\mathsf{R}\rightarrow\phi(C')[\phi(\mathsf{P})])|\mathsf{R}'$. This is matched by $C'[\mathsf{Q}]|\mathsf{R} \rightarrow \phi(C')[\phi(\mathsf{Q})])|\mathsf{R}'$, because by time-closure $(\phi(\mathsf{P}), \phi(\mathsf{Q})) \in \mathcal{R}$, hence $(\phi(C')[\phi(\mathsf{P})])|\mathsf{R}', \phi(C')[\phi(\mathsf{Q})])|\mathsf{R}') \in C[\mathcal{R}]$. Otherwise $C[\cdot]$ is not a reduction-context. This case is similar, except that we do not apply ϕ inside $\phi(C'[\cdot])$.
- $\underline{C'[\cdot] \rightarrow C''[\cdot] \text{ implies } C'[\mathsf{P}]|\mathsf{R} \rightarrow \mathsf{P}'}$. There are two subcases. If $C'[\cdot]$ is a reduction context, $C'[\mathsf{P}]|\mathsf{R} \rightarrow C''[\phi(\mathsf{P})]|\phi(\mathsf{R})$ is the reduction step. Then also $C'[\mathsf{Q}]|\mathsf{R} \rightarrow C''[\phi(\mathsf{Q})]|\phi(\mathsf{R})$. By $\mathcal{R}$'s time-closure we have

$$(C''[\phi(\mathsf{P})]|\phi(\mathsf{R}), C''[\phi(\mathsf{Q})]|\phi(\mathsf{R})) \in C[\mathcal{R}]$$

as required. Otherwise $C'[\cdot]$ is not a reduction context. This case is very similar, except that we do not apply ϕ inside $\phi(C'[\cdot])$.

$C'[\cdot]$ interacts with R. Again we have two subcases. If $C'[\cdot]$ is a reduction context, then our reduction is of the form $C'[\mathsf{P}]|\mathsf{R} \to (\nu\vec{x})(C''[\phi(\mathsf{P})]|\mathsf{R}')$ which is matched by $C'[\mathsf{Q}]|\mathsf{R} \to (\nu\vec{x})(C''[\phi(\mathsf{Q})]|\mathsf{R}')$, using time-closure. The other subcase, where $C'[\cdot]$ is not a reduction-context is similar.

$C'[\cdot]$ is a reduction-context and P interacts with R. Then the reduction is

$$C'[\mathsf{P}]|\mathsf{R} \to (\nu\vec{y})(\phi(C')[\mathsf{P}']|\mathsf{R}').$$

We proceed by induction on the number of restrictions guarding the hole in $C'[\cdot]$. In the base case, $C'[\cdot] = \mathsf{S}|[\cdot]$. Thus the reduction is (up to $\equiv$): $\mathsf{S}|\mathsf{P}|\mathsf{R} \to \phi(\mathsf{S})|(\nu\vec{x})(\mathsf{P}'|\mathsf{R}')$. As $\mathcal{R}$ is $|$-closed, we know that $(\mathsf{P}|\mathsf{R}, \mathsf{Q}|\mathsf{R}) \in \mathcal{R},$. Hence $C[\mathsf{P}] \equiv C'[\mathsf{P}|\mathsf{R}] \to \mathsf{P}'$. Using the outermost (IH) we can find a reduction $C'[\mathsf{Q}]|\mathsf{R} \to \mathsf{Q}'$ with $\mathcal{T} \vdash \phi(\mathsf{S})|(\nu\vec{x})(\mathsf{P}'|\mathsf{R}') = \mathsf{Q}'$, as required. For the inductive step, let $C'[\cdot] = (\nu x)C''[\cdot]$. As $C''[\cdot]$ must also be a reduction-context, our reduction is of the form, up to $\equiv$, $(\nu x)(C''[\mathsf{P}]|\mathsf{R}) \to (\nu x\vec{y})(\phi(C'')[\mathsf{P}']|\mathsf{R}')$. It is easy to see that then also $C''[\mathsf{P}]|\mathsf{R} \to (\nu\vec{y})(\phi(C'')[\mathsf{P}']|\mathsf{R}')$. The innermost (IH) then guarantees the existence of a process Q' such that $C''[\mathsf{Q}]|\mathsf{R} \to \mathsf{Q}'$ with

$$((\nu\vec{y})(\phi(C'')[\mathsf{P}']|\mathsf{R}'), \mathsf{Q}') \in C[\mathcal{R}].$$

Then also $C[\mathsf{Q}] \to (\nu x)\mathsf{Q}'$ and by definition of $C[\cdot]$:

$$\mathcal{T} \vdash (\nu x\vec{y})(\phi(C'')[\mathsf{P}']|\mathsf{R}') = (\nu x)\mathsf{Q}'.$$

The reduction was inferred using (Cong) as last rule. This case is immediate by the middle (IH), because $C[\mathcal{R}]$ is closed under $\equiv$.

The reduction was inferred using (Idle) as last rule. The we can also apply (Idle) to $C[\mathsf{Q}]$ and use time-closure, as above.

$C[\cdot] = (\nu x)C'[\cdot]$. This case is easy by the (IH).

$C[\cdot] = (!)x(\vec{v}).C'[\cdot]$. This case is trivial as the only applicable reduction is that induced by (Idle), though not necessarily as last rule. Since $C[\mathsf{P}]$ has no active timers we can match $C[\mathsf{P}] \to C[\mathsf{P}]$ by $C[\mathsf{Q}] \to C[\mathsf{Q}]$.

$C[\cdot] = \mathsf{timer}^1(\mathsf{R}, C'[\cdot])$. Again, all the process can do is idle, so, up to $\equiv$, the reduction step must be $\mathsf{timer}^1(\mathsf{R}, C'[\mathsf{P}]) \to C'[\mathsf{P}]$, clearly matched by $\mathsf{timer}^1(\mathsf{R}, C'[\mathsf{Q}]) \to C'[\mathsf{Q}]$, using the (IH).

$C[\cdot] = \mathsf{timer}^{t+1}(\mathsf{R}, C'[\cdot])$. This case is similar to the last.

$\underline{C[\cdot] = \mathsf{timer}^1(C'[\cdot], \mathsf{R})}$ Up to $\equiv$, all $C[\mathsf{P}]$ can do is idle. So $\mathsf{timer}^1(C'[\mathsf{P}], \mathsf{R}) \rightarrow \mathsf{R}$ is matched by $\mathsf{timer}^1(C'[\mathsf{Q}], \mathsf{R}) \rightarrow \mathsf{R}$ because $\mathsf{id} \subseteq \equiv \mathcal{R}$.

$\underline{C[\cdot] = \mathsf{timer}^{t+1}(C'[\cdot], \mathsf{R})}$. Easy using the (IH).

□

The question is now, given $\mathcal{R}$ when is $C^*[\mathcal{R}]$ a sound theory? We would like to say something like: if $\mathcal{R}$ is strongly reduction-closed, then $(C[\mathsf{ins}(\mathsf{sc}(\mathsf{sym}(\mathcal{R})))])^*$ is a sound theory. Unfortunately, things do not seem to be that simple. The next proposition shows what we can easily establish.

PROPOSITION 6 *Let $\mathcal{R}$ be a binary relation on π_t processes. If it contains $\equiv$, identifies all insensitive processes, is symmetric, time-closed, $|$-closed strongly reduction-closed and preserves strong barbs then $C^*[\mathcal{R}]$ is a sound theory.*

PROOF: By Lemma 17. □

We are now ready for the first decomposition theorem. It gives a sufficient condition for legitimately taking off restrictions.

THEOREM 19 *Let $\mathcal{T}$ be sound and assume x is a name. If $\mathcal{T} \vdash (\nu x)(\mathsf{P}|\mathsf{R}) = (\nu x)(\mathsf{Q}|\mathsf{R})$ for all R, then $\mathcal{T}_{\max} \vdash \mathsf{P} = \mathsf{Q}$.*

Rather than prove it directly, we consider a more general second Decomposition Theorem that implies the first, because the latter is not strong enough for our purposes: we would like know if scope extending transitions lead out of $\mathcal{T}_{max}$. This will be crucial later for the labelled characterisation of $\mathcal{T}_{max}$. Ideally, we would like to be sure that

$$\mathcal{T} \vdash (\nu\vec{y})(\overline{x}\langle\vec{z}\rangle|\mathsf{P}) = (\nu\vec{y})(\overline{x}\langle\vec{z}\rangle|\mathsf{Q}) \qquad \text{implies} \qquad \mathcal{T}_{max} \vdash \phi(\mathsf{P}) = \phi(\mathsf{Q})$$

where $\mathcal{T}$ be sound and $x \notin \{\vec{y}\}$, or, slight less conveniently,

$$\forall \mathsf{R}.\mathcal{T} \vdash (\nu\vec{y})(\mathsf{P}|\mathsf{R}|\mathsf{S}) = (\nu\vec{y})(\mathsf{Q}|\mathsf{R}|\mathsf{S}) \qquad \text{implies} \qquad \mathcal{T}_{max} \vdash \phi(\mathsf{P}) = \phi(\mathsf{Q}).$$

where S is some replication-free process, fixed for any choice of P and Q. Unfortunately, we have not been able to verify or falsify either. Venturing a guess, the latter decomposition is probably true, while the former might be false. Anyway, instead, we have to make do with the following theorem.

THEOREM 20 *Let $\mathcal{T}$ be sound. Let S be a process. Write $\mathsf{delay}^t(\mathsf{S})$ for the process $(\nu a)\mathsf{timer}^t(a.\mathsf{0}, \mathsf{S})$ where $a \notin \mathsf{fn}(\mathsf{S})$. Then*

$$\forall \mathsf{R}.\ \mathcal{T} \vdash (\nu\vec{y})(\mathsf{P} \mid \mathsf{R} \mid \mathsf{delay}^t(\mathsf{S})) = (\nu\vec{y})(\mathsf{Q} \mid \mathsf{R} \mid \mathsf{delay}^t(\mathsf{S}))$$

implies

$$\mathcal{T}_{\max} \vdash \mathsf{P} = \mathsf{Q}.$$

PROOF: Define

$$\mathcal{R} = \{(\mathsf{P}, \mathsf{Q}) \,|\, \mathcal{T} \vdash (\nu\vec{y})(\mathsf{P} \mid \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S})) = (\nu\vec{y})(\mathsf{Q} \,|\, \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S}))\}.$$

It is clear that $\mathcal{R}$ is symmetric and $|$-closed. Congruency of $\mathcal{T}$ implies that $\mathcal{R}$ contains $\equiv$ (because $\equiv \subseteq \mathcal{T}$) and identifies all insensitive terms. For strong reduction-closure, let $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ and $\mathsf{P} \to \mathsf{P}'$. Choose arbitrary R and $t > 0$. Then

$$(\nu\vec{y})(\mathsf{P} \mid \mathsf{delay}^1(\mathsf{R}) \,|\, \mathsf{delay}^{t+1}(\mathsf{S})) \to (\nu\vec{y})(\mathsf{P}' \mid \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S})) \stackrel{def}{=} \mathsf{P}''.$$

Using $\mathcal{T}$'s strong reduction-closure, we can find a reduction step

$$(\nu\vec{y})(\mathsf{Q} \mid \mathsf{delay}^1(\mathsf{R}) \,|\, \mathsf{delay}^{t+1}(\mathsf{S})) \to \mathsf{Q}''$$

with $\mathcal{T} \vdash \mathsf{P}'' = \mathsf{Q}''$. By induction on the derivation of this last reduction it is straightforward to see that there are exactly the following four possibilities.

- $\mathsf{Q} \to \mathsf{Q}'$ and $\mathsf{Q}'' = (\nu\vec{y})(\mathsf{Q}' \mid \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S}))$.
- $\mathsf{delay}^1(\mathsf{R}) \,|\, \mathsf{delay}^{t+1}(\mathsf{S}) \to \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S})$. Then $\mathsf{Q}'' = (\nu\vec{y})(\phi(\mathsf{Q}) \mid \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S}))$.
- The reduction was inferred using (IDLE) as last step. This means that Q'' is exactly as in the last case.
- The reduction was inferred using (CONG) as last step.

In all cases (by (IH) in the last) there is a reduction $\mathsf{Q} \to \mathsf{Q}'$ such that $\mathsf{Q}'' = (\nu\vec{y})(\mathsf{Q}' \mid \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S}))$, hence $(\mathsf{P}', \mathsf{Q}') \in \mathcal{R}$. This means that $\mathcal{R}$ is strongly reduction-closed.

For time-closure, choose once again an arbitrary R and $t > 0$ and set

$$\mathsf{T} = (\nu a)(\overline{a} \,|\, \mathsf{timer}^1(a, \overline{b})),$$

where a and b are fresh and distinct. Then

$$\mathcal{T} \vdash \mathsf{T} \,|\, (\nu\vec{y})(\mathsf{P} \mid \mathsf{delay}^1(\mathsf{R}) \,|\, \mathsf{delay}^{t+1}(\mathsf{S})) = \mathsf{T} \,|\, (\nu\vec{y})(\mathsf{Q} \mid \mathsf{delay}^1(\mathsf{R}) \,|\, \mathsf{delay}^{t+1}(\mathsf{S}))$$

But

$$\mathsf{T} \,|\, (\nu\vec{y})(\mathsf{P} \mid \mathsf{delay}^1(\mathsf{R}) \,|\, \mathsf{delay}^{t+1}(\mathsf{S})) \to \mathsf{0} \,|\, (\nu\vec{y})(\phi(\mathsf{P}) \mid \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S}))$$

By strong reduction closure we can find a matching transition

$$\mathsf{T} \,|\, (\nu\vec{y})(\mathsf{Q} \mid \mathsf{delay}^1(\mathsf{R}) \,|\, \mathsf{delay}^{t+1}(\mathsf{S})) \to \mathsf{U}.$$

Knowing that $\mathsf{0} \,|\, (\nu\vec{y})(\phi(\mathsf{P}) \mid \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S}))$ does not have a barb at b, we use Proposition 2 to infer $\mathsf{U} \not\downarrow_b$. By an easy induction on the derivation of the last reduction step we see that this means that

$$\mathsf{U} \equiv \mathsf{0} \,|\, (\nu\vec{y})(\phi(\mathsf{Q}) \mid \mathsf{R} \,|\, \mathsf{delay}^t(\mathsf{S})).$$

Then, as required

$$\mathcal{T} \vdash (\nu\vec{y})(\phi(\mathsf{P}) \mid \mathsf{R} \mid \mathsf{delay}^t(\mathsf{S})) = (\nu\vec{y})(\phi(\mathsf{Q}) \mid \mathsf{R} \mid \mathsf{delay}^t(\mathsf{S})),$$

which implies $\mathcal{R}$'s time-closure. It remains to establish strong-barb-preservation. Let $(\mathsf{P},\mathsf{Q}) \in \mathcal{R}$, $\mathsf{P}\downarrow_x$ but $\mathsf{Q}\not\downarrow_x$. Set $\mathsf{R} = \mathsf{timer}^1(x(\vec{v}).\overline{a},\mathsf{0})$, where a is fresh. Then

$$(\nu\vec{y})(\mathsf{P} \mid \mathsf{R} \mid \mathsf{delay}^{t+1}(\mathsf{S})) \rightarrow \mathsf{P}'\downarrow_a$$

but whenever

$$(\nu\vec{y})(\mathsf{Q} \mid \mathsf{R} \mid \mathsf{delay}^{t+1}(\mathsf{S})) \rightarrow \mathsf{Q}'$$

then $\mathsf{Q}'\not\downarrow_a$. In the light of Proposition 2, this violates $\mathcal{T}$'s soundness. □

Looking over the proof one may notice similarities in the verifications of time-closure and strong reduction-closure. It is in fact possible to show that the latter implies the former, under suitable side conditions, but we do not pursue this matter here. It is also easy to prove that Theorem 20 implies Theorem 19, essentially by noting that we can assume $S = \mathsf{0}$.

4.5 Reduction-Based Characterisations of $\mathcal{T}_{max}$

The existence of the maximum sound theory is a pleasant fact, but not especially useful on its own. To prove that processes are equated by $\mathcal{T}_{max}$, we need more tractable tools. In the rest of this chapter we will develop some of them. We start by comparing reduction congruence with barbed equivalences.

4.5.1 $\mathcal{T}_{max} = \overset{rc}{\sim} = \overset{rc}{\approx}$

We start our investigation of $\mathcal{T}_{max}$ by showing that it coincides with the two congruences introduced in §4.3.1.

THEOREM 21 $|\,\mathcal{T}_{\max}\,| = \mathcal{T}_{\max} = \overset{rc}{\sim} = \overset{rc}{\approx}$.

PROOF: First we show that $\overset{rc}{\sim}$ and $\overset{rc}{\approx}$ are sound theories. By definition they are π_t-congruences. Reduction-closure of $\overset{rc}{\approx}$, too, is by definition, whereas for $\overset{rc}{\sim}$ we need the help of Proposition 3. Identification of all insensitive terms is immediate because

$$\{(\mathsf{P},\mathsf{Q}) \mid \mathsf{P},\mathsf{Q} \text{ insensitive}\}$$

is clearly a strong barbed bisimulation and hence a barbed bisimulation. Consistency follows from $\mathsf{0} \overset{rc}{\not\approx} \overline{x}$. This gives $\overset{rc}{\sim}, \overset{rc}{\approx} \subseteq |\,\mathcal{T}_{max}\,|$. Conversely, by Proposition 3, $\mathcal{T}_{max}$ is strongly reduction closed and preservation of strong barbs follows form Proposition 2. Hence $|\,\mathcal{T}_{max}\,| \subseteq \overset{rc}{\sim} \subseteq \overset{rc}{\approx}$. Finally, $\mathcal{T}_{max} = |\,\mathcal{T}_{max}\,|$ is immediate from maximality and Lemma 9.1 □

Theorem 21 also justifies our choice of barb.

There is no Maximum Consistent, Reduction-Closed π_t-Theory

Now that we have a more concrete handle on what $\mathcal{T}_{max}$ looks like, we can fill in the missing part of the proof of Theorem 18.

LEMMA 18 *Let* I *be an insensitive term that is not equated with* $\mathsf{0}$ *by* $\equiv$*, for example* $(\nu x)\overline{x}$*. Define*

$$\mathcal{T} = \{(\mathsf{P}|\mathsf{I}, \mathsf{Q}|\mathsf{I}) \mid \mathsf{P}, \mathsf{Q} \text{ are } \pi_t \text{ processes}\}.$$

Then $\mathcal{T}$ *is consistent and reduction-closed.*

PROOF: We first show that $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ iff $\mathsf{P} \equiv \mathsf{Q}$ or for some context $C[\cdot]$ and some processes P_1, Q_1: $\mathsf{P} = C[\mathsf{P}_1|\mathsf{I}]$ and $\mathsf{Q} = C[\mathsf{Q}_1|\mathsf{I}]$. ($\Rightarrow$) follows by an easy induction on the derivation of $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$. For ($\Leftarrow$) we use an equally straightforward induction on the structure or $C[\cdot]$. Consequently we have $\mathcal{T} \not\vdash \overline{a} = \overline{b}$ whenever $a \neq b$ and $\mathcal{T}$ must be consistent. As to reduction closure, let $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \to \mathsf{P}'$. If $\mathsf{P} \equiv \mathsf{Q}$, then $\mathsf{Q} \to \mathsf{P}'$ is a matching transition because $\mathcal{T} \vdash \mathsf{P}' = \mathsf{P}'$. Otherwise $\mathsf{P} = C[\mathsf{P}_1|\mathsf{I}]$ and $\mathsf{Q} = C[\mathsf{Q}_1|\mathsf{I}]$. By induction on the structure of $C[\cdot]$ we construct the matching transition $\mathsf{Q} \to \mathsf{Q}'$. If $C[\cdot] = [\cdot]$ then $\mathsf{P}' = \mathsf{P}'_1|\mathsf{I}$ and $\mathsf{Q} \twoheadrightarrow \mathsf{Q}$ does the job. If $C[\cdot] = \mathsf{R}|C'[\cdot]$ where $C'[\cdot]$ is some reduction-context, we proceed by induction on the derivation of $\mathsf{P} \to \mathsf{P}'$. If $\mathsf{R} \to \mathsf{R}'$ and $\mathsf{P} \to \mathsf{R}'|\phi(C'[\mathsf{I}])$, then $\mathsf{Q} \to \mathsf{R}'|\phi(C'[\mathsf{I}])$ matches. We omit the remaining cases as they are similarly straightforward. □

We can now prove Theorem 18.2. Assume there was a maximum consistent and reduction-closed π_t-theory $\mathcal{T}$. By Theorem 21, $\overset{rc}{\approx}$ is also consistent and reduction-closed, hence $\overset{rc}{\approx} \subseteq \mathcal{T}$. But then $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ for all P and Q because $\mathsf{P}|\mathsf{I} \overset{rc}{\approx} \mathsf{P}$ for all P (Example 1), making $\mathcal{T}$ inconsistent, in violation of our assumption.

Alternative Barbs

As often with asynchronous π-calculi, the details of the choice of barbs do not always affect the resulting congruence and Theorem 21 suggests that the barbs introduced in Definition 18 are no exception. One feature of that definition that may appear unnatural in the context of a timed calculus is that they do not consider time a first-class citizen in the sense that one can only use them to directly specify what a process can emit on a given name, either immediately or eventually, but not when that emission ought to take place. The definition below proposes a barb that specifies a temporal interval in addition to a name.

DEFINITION 30 Assume x is a name and $m, n \in \mathbb{N}$. We write

$$\mathsf{P} \Downarrow_{x[m,n]} \qquad \text{iff} \qquad \mathsf{P} \underbrace{\to \cdots \to}_{i} \mathsf{Q} \downarrow_x, m \leq i \leq n.$$

Now let $\overset{rc'}{\approx}$ be the congruence induced, as in Definition 18, by the set of all barbs $\Downarrow_{x[m,n]}$. Let $\overset{rc''}{\approx}$ be the congruence generated by barbs of the form $\Downarrow_{x[n,n]}$.

Not surprisingly, the characterisation of $\mathcal{T}_{max}$ as largest barbed congruence remains unaffected by this change of barb.

THEOREM 22 $\mathcal{T}_{\max} = \overset{rc'}{\approx} = \overset{rc''}{\approx}$.

PROOF: Straightforward with judicious use of reduction-spotters, after noting that $\mathsf{P}\downarrow_x$ iff $\mathsf{P}\Downarrow_{x[0,0]}$. □

4.5.2 A Behavioural Characterisation of $\mathcal{T}_{max}$

In this section we provide a behavioural characterisation of $\mathcal{T}_{max}$, along the lines of Chapter 10.4 in [74] and Proposition 3.24 of [61]. The required set-theory is altogether standard and can be found in introductory text books on the subject such as [63, 66, 80].

DEFINITION 31 Let P be a process. We write $\mathsf{P} \rightsquigarrow$ to indicate that $\mathsf{P}\downarrow_x$ for some name x. Let κ be an ordinal. We define $\simeq_\kappa$ by transfinite induction.

- $\mathsf{P} \simeq_0 \mathsf{Q}$ if for all $C[\cdot]$: $C[\mathsf{P}] \rightsquigarrow \Leftrightarrow\ C[\mathsf{Q}] \rightsquigarrow$.
- $\mathsf{P} \simeq_{\kappa+1} \mathsf{Q}$ if for all $C[\cdot]$:

$$\begin{aligned} C[\mathsf{P}] \to \mathsf{P}' \quad &\Rightarrow \quad \exists \mathsf{Q}'. C[\mathsf{Q}] \to \mathsf{Q}', \mathsf{P}' \simeq_\kappa \mathsf{Q}', \\ C[\mathsf{Q}] \to \mathsf{Q}' \quad &\Rightarrow \quad \exists \mathsf{P}'. C[\mathsf{P}] \to \mathsf{P}', \mathsf{P}' \simeq_\kappa \mathsf{Q}', \end{aligned}$$

- For all limit ordinals λ: $\mathsf{P} \simeq_\lambda \mathsf{Q}$ if for all $\kappa < \lambda : \mathsf{P} \simeq_\kappa \mathsf{Q}$.

Then we set: $\simeq\ = \bigcap_\kappa \simeq_\kappa$.

The rest of this section will establish that $\simeq\ = \mathcal{T}_{max}$.

LEMMA 19 *Let $\kappa > 0$ be an ordinal. If $\mathsf{P} \simeq_\kappa \mathsf{Q}$ and $C[\mathsf{P}]\downarrow_x$ then $C[\mathsf{Q}]\downarrow_x$.*

PROOF: By transfinite induction. The only non-trivial case is that of successor ordinals. Assume $\mathsf{P} \simeq_{\kappa+1} \mathsf{Q}$ and $C[\mathsf{P}]\downarrow_x$. Define

$$C'[\cdot] = (\nu \vec{y})([\cdot] \,|\, \mathsf{timer}^1(x(\vec{v}).\overline{a}, \mathsf{0}))$$

where $\mathsf{fn}(C[\mathsf{P}]), \mathsf{fn}(C[\mathsf{P}]) \subseteq \{\vec{y}\}$ and a is fresh. Then by Lemma 8.5: $C'[C[\mathsf{P}]] \to \mathsf{P}'\downarrow_a$. Hence there must be a reduction $C'[C[\mathsf{Q}]] \to \mathsf{Q}'$ where $\mathsf{P}' \simeq_\kappa \mathsf{Q}'$. By (IH): $\mathsf{Q}'\downarrow_a$. Using Lemma 8.5 again, we conclude $C[\mathsf{Q}]\downarrow_x$. □

LEMMA 20 *Let κ_0 be an ordinal. Assume that $\mathsf{P} \simeq_0 \mathsf{Q}$ and for all successor ordinals $\kappa' \leq \kappa_0$: $\mathsf{P} \simeq_{\kappa'} \mathsf{Q}$. Then $\mathsf{P} \simeq_\kappa \mathsf{Q}$ for all ordinals $\kappa \leq \kappa_0$.*

PROOF: By straightforward transfinite induction on κ_0. □

LEMMA 21 *[Anti-Monotonicity of $\simeq_\kappa$] If $\alpha \leq \beta$ then $\simeq_\beta \subseteq \simeq_\alpha$.*

PROOF: By transfinite induction on β. For successor ordinals, assume $\mathsf{P} \simeq_{\beta+1} \mathsf{Q}$. Let $C[\mathsf{P}] \to \mathsf{P}'$. Then we can find $C[\mathsf{Q}] \to \mathsf{Q}' \simeq_\beta \mathsf{P}'$ and we can use the (IH) to conclude to $\mathsf{P}' \simeq_\alpha \mathsf{Q}'$ and thus also to $\mathsf{P} \simeq_{\alpha+1} \mathsf{Q}$ for all $\alpha \leq \beta$. Now assume $C[\mathsf{P}]\downarrow_x$. Then by Lemma 19: $C[\mathsf{Q}]\downarrow_x$, but then also $\mathsf{P} \simeq_0 \mathsf{Q}$. Hence $\mathsf{P} \simeq_\alpha \mathsf{Q}$ for all $\alpha \leq \beta$ such that α is not a limit ordinal. Now apply Lemma 20. □

LEMMA 22 *For all ordinals $\kappa > 0$: $\simeq_\kappa$ is a π_t-congruence.*

PROOF: By transfinite induction we show that (1) for all $\kappa \geq 0$: $\simeq_\kappa$ is transitive as well as symmetric; and (2) for all ordinals $\kappa > 0$: $\simeq_\kappa$ is a congruence.

Clearly $\simeq_0$ is a transitive and symmetric relation containing $\equiv$. Now assume $\mathsf{P} \simeq_{\kappa+1} \mathsf{Q}$. Clearly $\simeq_{\kappa+1}$ contains $\equiv$ and is symmetric. By (IH) it is also transitive. For congruency, we use contexts' being closed under composition. Limit ordinals are immediate. □

LEMMA 23 *$\simeq$ is reduction-closed.*

PROOF: Assume $\mathsf{P} \simeq \mathsf{Q}$ and $\mathsf{P} \to \mathsf{P}'$. Using anti-monotonicity we know that

$$\forall\kappa.\exists\mathsf{Q}'.\forall\kappa' \leq \kappa.\mathsf{Q} \to \mathsf{Q}', \mathsf{P}' \simeq_\kappa \mathsf{Q}'. \tag{4.1}$$

Assume that

$$\forall\mathsf{Q}'.\exists\kappa_{\mathsf{Q}'}.(\mathsf{Q} \to \mathsf{Q}' \Rightarrow \mathsf{P}' \not\simeq_{\kappa_{\mathsf{Q}'}} \mathsf{Q}').$$

Then we can construct a map f from appropriate processes to ordinals by setting

$$\mathsf{Q}' \mapsto \kappa_{\mathsf{Q}'}$$

whenever $\mathsf{Q} \to \mathsf{Q}'$. As there are at most countably many such processes Q' we can construct an ordinal κ^+ such that $f(\mathsf{Q}') < \kappa^+$ for all appropriate Q', in contradiction to (4.1). Hence

$$\exists\mathsf{Q}'.\forall\kappa.(\mathsf{Q} \to \mathsf{Q}' \;\Rightarrow\; \mathsf{P}' \simeq_\kappa \mathsf{Q}').$$

This means $\mathsf{P}' \simeq \mathsf{Q}'$. Hence $\simeq$ must be reduction-closed (using Proposition 3). □

THEOREM 23 *(Behavioural Characterisation of $\mathcal{T}_{\max}$)*

$$\simeq \;=\; \mathcal{T}_{\max}.$$

PROOF: First we show that $\mathcal{T}_{max} \subseteq \simeq_\kappa$ by transfinite induction. The base case is immediate from Proposition 2. The inductive steps are straightforward from the (IH) and the definitions. Hence $\mathcal{T}_{max} \subseteq \simeq$.

For the reverse inclusion we establish $\simeq$ to be a sound theory. Consistency is immediate because $\mathsf{0} \not\simeq_0 \overline{x}$, hence $\mathsf{0} \not\simeq \overline{x}$. Congruency is by Lemma 22. Reduction-closure is guaranteed by Lemma 23 and the identification of all insensitive terms follows from $\mathcal{T}_{max} \subseteq \simeq$. □

A Variant

The base case in Definition 31 requires $\simeq_0$ to be a congruence. For our later development it will be convenient to be more liberal and let congruency emerge in the course of the transfinite induction rather than requiring it from the start. We show now that this results in the same fixpoint, $\mathcal{T}_{max}$.

DEFINITION 32 Let κ be an ordinal. We define $\simeq'_\kappa$ by transfinite induction.

- $\mathsf{P} \simeq'_0 \mathsf{Q}$ if $\mathsf{P} \rightsquigarrow \Leftrightarrow \mathsf{Q} \rightsquigarrow$.
- $\mathsf{P} \simeq'_{\kappa+1} \mathsf{Q}$ if for all $C[\cdot]$:

$$\begin{aligned} C[\mathsf{P}] \to \mathsf{P}' &\Rightarrow \exists \mathsf{Q}'. C[\mathsf{Q}] \to \mathsf{Q}', \mathsf{P}' \simeq'_\kappa \mathsf{Q}', \\ C[\mathsf{Q}] \to \mathsf{Q}' &\Rightarrow \exists \mathsf{P}'. C[\mathsf{P}] \to \mathsf{P}', \mathsf{P}' \simeq'_\kappa \mathsf{Q}', \end{aligned}$$

- For all limit ordinals λ: $\mathsf{P} \simeq'_\lambda \mathsf{Q}$ if for all $\kappa < \lambda : \mathsf{P} \simeq'_\kappa \mathsf{Q}$.

Then we set: $\simeq' = \bigcap_\kappa \simeq'_\kappa$.

LEMMA 24 *1. (Cf. Lemma 19) Let $\kappa > 0$ be an ordinal. If $\mathsf{P} \simeq'_\kappa \mathsf{Q}$ and $C[\mathsf{P}]\downarrow_x$ then $C[\mathsf{Q}]\downarrow_x$.*

2. (Cf. Lemma 20) Let κ_0 be an ordinal. Assume that $\mathsf{P} \simeq'_0 \mathsf{Q}$ and for all successor ordinals $\kappa' \leq \kappa_0$: $\mathsf{P} \simeq'_{\kappa'} \mathsf{Q}$. Then $\mathsf{P} \simeq'_\kappa \mathsf{Q}$ for all ordinals $\kappa \leq \kappa_0$.

3. (Cf. Lemma 21) If $\alpha \leq \beta$ then $\simeq_\beta \subseteq \simeq_\alpha$.

4. $\simeq_\kappa \subseteq \simeq'_\kappa$.

5. $\simeq'_{\kappa+2} \subseteq \simeq_\kappa$.

6. For all limit ordinals λ: $\simeq'_\lambda = \simeq_\lambda$.

PROOF: Items (1), (2) and (3) are proved exactly as their counterparts in Section 4.5.2. For (4) and (5) we proceed by transfinite induction on κ. The former is straightforward, for the latter, if $\kappa = 0$, then $\simeq'_2 \subseteq \simeq'_1 \subseteq \simeq_0$ by anti-monotonicity

and (4). Next, assume $\kappa = \alpha + 1$. If $C[\mathsf{P}] \to \mathsf{P}'$ then there is a reduction step $C[\mathsf{Q}] \to \mathsf{Q}'$ with $\mathsf{P}' \simeq'_{\alpha+2} \mathsf{Q}'$. By (IH) then $\mathsf{P}' \simeq'_{\alpha} \mathsf{Q}'$, hence $\mathsf{P} \simeq_{\alpha+1}$ i.e. $\mathsf{P} \simeq_{\kappa} \mathsf{Q}$. For limit ordinals λ: $\mathsf{P} \simeq'_{\lambda} \mathsf{Q}$ implies for all $\kappa < \lambda$: $\mathsf{P} \simeq'_{\kappa} \mathsf{Q}$, so for all $\kappa < \lambda$: $\mathsf{P} \simeq'_{\kappa+2} \mathsf{Q}$. But then by (IH) for all $\kappa < \lambda$: $\mathsf{P} \simeq_{\kappa} \mathsf{Q}$, which means $\mathsf{P} \simeq_{\lambda} \mathsf{Q}$. Finally, (6) is easy from the definition and (4) and (5) □

PROPOSITION 7 $\simeq' = \simeq$.

PROOF: Immediate from Lemma 24. □

4.5.3 A Modal Characterisation of $\mathcal{T}_{max}$

In [50] Hennessy and Milner propose an infinitary modal logic that allows to specify properties of processes logically rather than by (co-)algebraic means. Honouring the originators, such logics are called *Hennessy-Milner logics*. One of their salient features is that they pin down process properties by reference to the observations each possible environment may or must be able to make. This works fine as long as we have a convenient mathematical handle on those observations. In practise that means having labelled transitions. Unfortunately those are not always easily available. For advanced calculi it is often more convenient to specify reductions rather than transitions. Thus it may be a good idea to have a logic that allows to specify process properties with reference to reductions only. This section proposes *Context Logic*, an infinitary modal logic where elementary equivalence (in the standard model) coincides with $\mathcal{T}_{max}$. Unlike Hennessy-Milner logic, which employs observations as modalities, we choose *contexts*: if P is a process, $C[\cdot]$ a context and ψ a formula, then $\mathsf{P} \vDash \langle C \rangle \psi$ if and only if there is a reduction $C[\mathsf{P}] \to \mathsf{Q}$ such that $\mathsf{Q} \vDash \psi$. One could say that process properties are specified by letting contexts do the observation and talking about what context together with the process under observation may or must do. This internalises observations and obviates the need for labelled transitions. Well, almost: we need an atomic formula $\rightsquigarrow$ which can be seen as an observation that is not internalised, but $\rightsquigarrow$ can be defined solely in terms of reductions.

We do not claim that Context Logic is a particularly convenient tool. At this stage, its purpose is to inspire further development.

Infinitary Modal Logics

We begin by presenting the basic definition of modal logic, following [21].

DEFINITION 33 A *modal similarity type* is a pair $\tau = (O, \#)$ where O is a non-empty set of *modal operators* and $\#$ is a function from modal operators to $\mathbb{N}$. We let $\triangle$ range over elements of O and $\#\triangle$ denotes the *arity* of $\triangle$.

DEFINITION 34 Let κ be a cardinal and assume that A is a set of *atomic formulae*, ranged over by p. Then the set of *κ-infinitary modal formulae over τ and A* is inductively generated by the following grammar.

$$\phi ::= p \mid \neg\phi \mid \bigwedge \Phi \mid \triangle(\phi_1, ..., \phi_{\#\triangle})$$

Here Φ is a set of κ-infinitary modal formulae over τ and A such that $|\Phi| < \kappa$. For each n-ary modal operator $\triangle$ we define its *dual* ∇ as $\nabla(\phi_1, \ldots, \phi_n) = \neg\triangle(\neg\phi_1, \ldots, \neg\phi_n)$. We also frequently use these additional abbreviations.

$$\bigvee \Phi = \neg \bigwedge \neg\Phi \qquad \phi \to \psi = \neg\phi \vee \psi \qquad \bot = \bigwedge \emptyset \qquad \top = \neg\bot$$

The *modal depth* $\mathsf{md}(\phi)$ of a formula ϕ is inductively defined by the following clauses.

$$\begin{aligned} \mathsf{md}(\triangle(\phi_1, \ldots, \phi_{\#\triangle})) &= 1 + \mathsf{max}(\mathsf{md}(\phi_1), ..., \mathsf{md}(\phi_n)) \\ \mathsf{md}(\neg\phi) &= \mathsf{md}(\phi) \\ \mathsf{md}(\textstyle\bigwedge \Phi) &= \mathsf{sup}(\{\mathsf{md}(\phi) \mid \phi \in \Phi\}) \end{aligned}$$

The $\mathsf{sup}(\cdot)$ operation returns an ordinal.

Models of Infinitary Modal Logics

DEFINITION 35 Given a modal similarity type τ, a *τ-frame* is a tuple $\mathfrak{F}$ containing a non-empty set W and for each modal operator $\triangle$ a relation $R_\triangle \subseteq W^{n+1}$ where n is the arity of $\triangle$. The elements of W are called *states*, *points* or *processes*. A (τ, A)-*model* is a pair $\mathfrak{M} = (\mathfrak{F}, V)$ where $\mathfrak{F}$ is a τ-frame and $V : \mathsf{A} \to \mathcal{P}(W)$ is a *valuation*. The *satisfaction relation* $M \vDash_w \phi$ is defined inductively by the following rules.

$\mathfrak{M} \vDash_w \triangle(\phi_1, \ldots, \phi_n)$	iff	for some $w_1, \ldots, w_n \in W$ with $R_\triangle(w, w_1, \ldots, w_n)$ we have, for each i : $M \vDash_{w_i} \phi_i$
$\mathfrak{M} \vDash_w \bigwedge \Phi$	iff	$\mathfrak{M} \vDash_w \phi$ for all $\phi \in \Phi$
$\mathfrak{M} \vDash_w \neg\phi$	iff	$\mathfrak{M} \nvDash_w \phi$
$\mathfrak{M} \vDash_w \bot$	iff	false
$\mathfrak{M} \vDash_w p$	iff	$w \in V(p)$

We will usually write $w \vDash \phi$ for $\mathfrak{M} \vDash_w \phi$, hoping that no ambiguities arise. The set of all formulae ϕ such that $w \vDash \phi$ is denoted $\mathsf{thm}(w)$, and by $\mathsf{thm}^\kappa(w)$ we mean $\mathsf{thm}(w)$ restricted to formulae of modal depth not exceeding κ. A formula ϕ is *globally* or *universally true in a model* $\mathfrak{M}$, written $\mathfrak{M} \vDash \phi$ if it is satisfied at all points in $\mathfrak{M}$; ϕ is *satisfiable in* $\mathfrak{M}$ if there is some state in $\mathfrak{M}$ at which ϕ is true. A formula is *falsifiable* or *refutable in* $\mathfrak{M}$ if its negation is satisfiable in $\mathfrak{M}$. A set Σ of formulae is *globally true (satisfiable) in* $\mathfrak{M}$ if each of its members is.

A formula ϕ is *valid at a state w is a frame* $\mathfrak{F}$, notation $\mathfrak{F} \vDash_w \phi$, or just $w \vDash \phi$, if ϕ is true at w in every model $(\mathfrak{F}, V)$ based on $\mathfrak{F}$; ϕ is *valid in a frame* $\mathfrak{F}$, written $\mathfrak{F} \vDash \phi$ if it is valid at every state in $\mathfrak{F}$. A formula ϕ is *valid on a class* F, written $F \vDash \phi$ if it is valid on every frame $\mathfrak{F}$ in F. It is *valid*, written $\vDash \phi$, if it is valid on the class of all frames.

DEFINITION 36 Two processes P and Q inside a model $\mathfrak{M}$ are *elementarily equivalent*, written $\mathsf{P} =_{\mathrm{ee}} \mathsf{Q}$, if for all formulae $\phi : \mathsf{P} \vDash \phi \iff \mathsf{Q} \vDash \phi$. We say P and Q are *κ-elementarily equivalent*, written $\mathsf{P} =^{\kappa}_{\mathrm{ee}} \mathsf{Q}$, if for all formulae ϕ with $\mathsf{md}(\phi) \leq \kappa : \mathsf{P} \vDash \phi \iff \mathsf{Q} \vDash \phi$.

Context Logic

Now that we have assembled the basic tools of modal logics, we can start our programme of characterising $\mathcal{T}_{max}$ as elementary equivalence.

DEFINITION 37 *Context-Logic* is an ω_1-infinitary modal modal logic where the modal operators are built from π-calculus terms. More precisely, modal operators are of the form $\langle C \rangle$ where $C[\cdot]$ is a π_t-calculus context. Hence Context Logic has the following formulae.

$$\phi \ ::= \ \langle C \rangle \phi \,|\, \neg\phi \,|\, \bigwedge \Phi \,|\, \rightsquigarrow$$

Here $\rightsquigarrow$ is the only propositional constant.

Deviating somewhat for the sake of simplicity from the definitions in the previous section, a model for a Context Logic is a tuple $\mathfrak{M} = (M, \rhd^{\mathfrak{M}}, \{C^{\mathfrak{M}}\}_{C[\cdot]}, \rightsquigarrow^{\mathfrak{M}})$ where $\rhd^{\mathfrak{M}}$ is a binary relation on M, called *reduction*, each $C_i^{\mathfrak{M}}$ is a unary function on M, collectively called *contexts* and $\rightsquigarrow_i^{\mathfrak{M}}$ is a unary predicate on M, called *may-barb*. Furthermore, the satisfaction relation $\mathsf{P} \vDash \phi$ is given inductively by the following clauses.

$$\begin{array}{lll}
\mathsf{P} \vDash \langle C \rangle \phi & \text{iff} & \mathsf{Q} \vDash \phi \text{ for some } \mathsf{Q} \text{ such that } C^{\mathfrak{M}}[\mathsf{P}] \rhd^{\mathfrak{M}} \mathsf{Q} \\
\mathsf{P} \vDash \neg\phi & \text{iff} & \mathsf{P} \nvDash \phi \\
\mathsf{P} \vDash \bigwedge \Phi & \text{iff} & \mathsf{P} \vDash \phi \ \text{ for all } \phi \in \Phi \\
\mathsf{P} \vDash \rightsquigarrow & \text{iff} & \mathsf{P} \rightsquigarrow^{\mathfrak{M}}
\end{array}$$

$$\begin{array}{lll}
\mathsf{P} \vDash [C]\phi & \text{iff} & \mathsf{Q} \vDash \phi \text{ whenever } C^{\mathfrak{M}}[\mathsf{P}] \rhd^{\mathfrak{M}} \mathsf{Q} \\
\mathsf{P} \vDash \bot & \text{iff} & \text{false} \\
\mathsf{P} \vDash \top & \text{iff} & \text{true}
\end{array}$$

Reduction Congruence as Elementary Equivalence in π_t

We now show that $\mathcal{T}_{max}$ coincides with $=_{\mathrm{ee}}$ in the intended model of Context Logic. This model has π_t processes as points, uses the one-step reduction relation $\rightarrow$ as $\rhd^{\mathfrak{M}}$, interprets the $\langle C \rangle$ modality by $C[\cdot]$ and $\mathsf{P} \rightsquigarrow^{\mathfrak{M}}$ iff for some name x: $\mathsf{P}\downarrow_x$.

LEMMA 25 *For all ordinals κ:* $\mathsf{P} =^{\kappa}_{ee} \mathsf{Q}$ *iff* $\mathsf{P} \simeq'_{\kappa} \mathsf{Q}$.

PROOF: By transfinite induction on κ. Assume $\mathsf{P} \not\simeq'_{\kappa} \mathsf{Q}$. We construct a formula ϕ with $\mathsf{md}(\phi) \leq \kappa$ such that $\mathsf{P} \models \phi$ but $\mathsf{Q} \not\models \phi$.

If $\kappa = 0$, then $\mathsf{P} \rightsquigarrow$ but $\mathsf{Q} \not\rightsquigarrow$, or vice versa. Hence $\rightsquigarrow$ is an appropriate formula of modal depth 0.

If $\kappa = \alpha + 1$, then some reduction $C[\mathsf{P}] \rightarrow \mathsf{P}'$ must exist such that whenever $C[\mathsf{Q}] \rightarrow \mathsf{Q}'$: $\mathsf{P}' \not\simeq_{\alpha} \mathsf{Q}'$. By (IH) some formula $\phi_{\mathsf{Q}'}$ with a modal depth not exceeding α can be found such that w.l.o.g. $\mathsf{P}' \models \phi_{\mathsf{Q}'}$ but $\mathsf{Q}' \not\models \phi_{\mathsf{Q}'}$. As there can be at most countably infinitely many reductions $C[\mathsf{Q}] \rightarrow \mathsf{Q}'$, the formula $\phi = \langle C \rangle \bigwedge \{\phi_{\mathsf{Q}'} \mid C[\mathsf{Q}] \rightarrow \mathsf{Q}'\}$ is valid with $\mathsf{md}(\phi) \leq \alpha + 1$. As clearly $\mathsf{P} \models \phi$ but $\mathsf{Q} \not\models \phi$, ϕ works as required.

If κ is a limit, $\mathsf{P} \neq^{\kappa}_{\mathrm{ee}} \mathsf{Q}$ is immediate from the (IH).

Conversely, let $\mathsf{P} \simeq'_{\kappa} \mathsf{Q}$ and assume that $\mathsf{P} \models \phi$.

If $\kappa = 0$, we proceed by straightforward induction on the structure of ϕ, noting that ϕ cannot have subformulae of the form $\langle C \rangle \phi'$.

For $\kappa = \alpha + 1$ we again employ induction on the structure of ϕ.

$\phi = \rightsquigarrow$, then by anti-monotonicity (24.2): $\mathsf{P} \simeq'_0 \mathsf{Q}$, hence $\mathsf{Q} \models \phi$.

$\phi = \bigwedge \Phi$. This is immediate by (IH).

$\phi = \langle C \rangle \phi'$. Then there is a process P' with $C[\mathsf{P}] \rightarrow \mathsf{P}'$ and $\mathsf{P}' \models \phi'$. This implies a reduction $C[\mathsf{Q}] \rightarrow \mathsf{Q}'$ and $\mathsf{P}' \simeq'_{\alpha} \mathsf{Q}'$, so by (IH): $\mathsf{Q}' \models \phi'$, which in turn means $\mathsf{Q} \models \phi$.

κ is a limit. This is again immediate by (IH).

□

THEOREM 24 *[74]* $\mathsf{P} =_{ee} \mathsf{Q}$ *iff* $\mathcal{T}_{\max} \vdash \mathsf{P} = \mathsf{Q}$.

PROOF: Immediate from Lemma 25 and the coincidence of $\simeq$ with $\simeq'$ (Proposition 7). □

In [110], Shelah characterised the elementary equivalence relation of first-order logic purely algebraically via ultraproducts. It would be interesting to see if the same can be done for $=_{\mathrm{ee}}$.

A Simple Example of Using Context Logic

For a simple example of using Context Logic, define

$$\begin{aligned} C^a_S[\cdot] &= (\nu S)([\cdot] \,|\, \mathsf{timer}^1(x(\vec{v}).\overline{a}, 0)) \\ \phi_0 &= \bigwedge_{a \notin S \subseteq_{fin} \mathcal{N}} [C^a_S] \\ \phi_{n+1} &= [[\cdot]]\phi_n. \end{aligned}$$

Then $\mathsf{P} \vDash \phi_n$ iff P can emit on x after n reductions and $\mathsf{P} \vDash \phi_7 \wedge \bigwedge_{n \neq 7} \neg\phi_n$ iff P can emit on x after 7 reductions and only then. Specifications in this bare form of Context Logic usually require infinite conjunctions even for simple properties and are hence of limited utility.

4.6 Transition-Based Characterisations of $\mathcal{T}_{max}$

All characterisations of $\mathcal{T}_{max}$ have so far been based on reductions. As repeatedly pointed out already, experience suggests that most substantial reasoning is easier with equalities based on labelled transitions. Much of process theory has consequently been a quest for suitable, that is sound, labelled approximations to prominent equivalences. In this section we present a labelled characterisation of $\mathcal{T}_{max}$. We begin by considering standard synchronous and asynchronous transitions systems and the associated (weak or strong) bisimilarities. It turns out that they are unsuitable since they fail to be congruences. To overcome these problems we then propose yet another transition system, which integrates asynchrony and time passing better than the previous candidates. With additional, but easily verified closure conditions, the associated strong bisimilarity turns out to be exactly $\mathcal{T}_{max}$, but proving this fact is not straightforward.

4.6.1 Synchronous and Asynchronous Bisimulations

Strong Synchronous Bisimulations

In Section 4.3.2 we defined a synchronous labelled transition relation $\xrightarrow{l}$ and we will now investigate how the induced (strong) bisimilarity relates to $\mathcal{T}_{max}$. The definitions of (strong) bisimulations and (strong) bisimilarity remain unchanged from Chapter 2. For convenience we use the synchronous transition system in Figure 4.3, which has exactly the same transitions, up to $\equiv$ as is readily established.

Example 2 Consider the following two processes

$$\mathsf{P}(\mathsf{R}) = (\nu a)(\overline{a} \,|\, \mathsf{timer}^1(a, \mathsf{R})) \qquad \mathsf{Q}(\mathsf{R}) = (\nu a)(\overline{a} \,|\, \mathsf{timer}^1(a.\mathsf{R}, 0)).$$

Here a is fresh.

$$\text{(Out)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_s \mathsf{0}}$$

$$\text{(In)}\quad \frac{}{x(\vec{v}).\mathsf{P} \xrightarrow{x(\vec{z})}_s \mathsf{P}\{\vec{z}/\vec{v}\}}$$

$$\text{(Rep)}\quad \frac{}{!x(\vec{v}).\mathsf{P} \xrightarrow{x(\vec{z})}_s !x(\vec{v}).\mathsf{P} \mid \mathsf{P}\{\vec{z}/\vec{v}\}}$$

$$\text{(TimeIn)}\quad \frac{}{\mathsf{timer}^{t+1}(x(\vec{v}).\mathsf{P}, Q) \xrightarrow{x(\vec{z})}_s \mathsf{P}\{\vec{z}/\vec{v}\}}$$

$$\text{(Par)}\quad \frac{\mathsf{P} \xrightarrow{l}_s \mathsf{P}' \qquad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{l}_s \mathsf{P}' \mid \phi(\mathsf{Q})}$$

$$\text{(Com)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid x(\vec{v}).\mathsf{P} \xrightarrow{\tau}_s \mathsf{P}\{\vec{y}/\vec{v}\}}$$

$$\text{(Rep)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid !x(\vec{v}).\mathsf{P} \xrightarrow{\tau}_s \mathsf{P}\{\vec{y}/\vec{v}\} \mid !x(\vec{v}).\mathsf{P}}$$

$$\text{(TimeIn}')\quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid \mathsf{timer}^{t}(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \xrightarrow{\tau}_s \mathsf{P}\{\vec{y}/\vec{v}\}}$$

$$\text{(Res)}\quad \frac{\mathsf{P} \xrightarrow{l}_s \mathsf{Q} \qquad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)}{(\nu x)\mathsf{P} \xrightarrow{l}_s (\nu x)\mathsf{Q}}$$

$$\text{(Open)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_s \mathsf{Q} \qquad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle}_s \mathsf{Q}}$$

$$\text{(Cong)}\quad \frac{\mathsf{P} \equiv \mathsf{P}' \qquad \mathsf{P}' \xrightarrow{l}_s \mathsf{Q}' \qquad \mathsf{Q}' \equiv \mathsf{Q}}{\mathsf{P} \xrightarrow{l}_s \mathsf{Q}}$$

$$\text{(Idle)}\quad \frac{}{\mathsf{P} \xrightarrow{\tau}_s \phi(\mathsf{P})}$$

Figure 4.3: An alternative account of synchronous transitions for the asynchronous timed π-calculus.

It is easy to see that $\mathsf{P}(\overline{x}) \sim \mathsf{Q}(\overline{x})$. But $\phi(\mathsf{P}(\overline{x})) \equiv \overline{x} \,|\, (\nu a)\overline{a}$ while $\phi(\mathsf{Q}(\overline{x})) \equiv (\nu a)\overline{a}$. Hence $\phi(\mathsf{P}(\overline{x})) \not\sim \phi(\mathsf{Q}(\overline{x}))$.[1] This means that

$$\overline{y} \,|\, \mathsf{P}(\overline{x}) \xrightarrow{\overline{y}} \overline{x} \,|\, (\nu a)\overline{a} \qquad \text{but} \qquad \overline{y} \,|\, \mathsf{Q}(\overline{x}) \xrightarrow{\overline{y}} (\nu a)\overline{a}.$$

So $\sim$ cannot be closed under parallel composition. Consequently, $\sim$ cannot coincide with $\mathcal{T}_{max}$. This is not the only shortcoming of $\sim$: we will later show (Example 3) that the identity forwarder fw_{xx} and $\mathsf{0}$ are equated by the largest sound theory, but they are clearly not related by synchronous bisimilarity, strong or otherwise, because fw_{xx} has a transition $\mathsf{fw}_{xx} \xrightarrow{x(\vec{y})}_s \mathsf{fw}_{xx} | \overline{x}\langle\vec{y}\rangle$ that $\mathsf{0}$ lacks.

Before moving to other equivalences, we summarise a few facts about $\sim$.

LEMMA 26 *1. If* $\mathsf{P} \sim \mathsf{Q}$ *then* $(\nu x)\mathsf{P} \sim (\nu x)\mathsf{Q}$.

2. If $\mathsf{P} \sim \mathsf{Q}$ *then* $\mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{P}) \sim \mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{Q})$.

3. $\sim$ *is not closed under parallel composition, renaming or timestepping.*

PROOF: For (1) the proof is completely standard, see [54]. For (2), let $\mathcal{R}$ be given by

$$\mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{P}) \;\; \mathcal{R} \;\; \mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{Q})$$

whenever $\mathsf{P} \sim \mathsf{Q}$. Clearly every transition by $\mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{P})$ is matched by one of $\mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{Q})$, so $\mathcal{R} \cup \sim$ is is a strong bisimulation. For (3), we have already presented a counterexample that shows the failure of congruency for parallel composition and application of ϕ. Lemma 34 presents two processes $\mathsf{P} \sim \mathsf{Q}$ such that $\mathsf{P}\{x/y\} \not\sim \mathsf{Q}\{x/y\}$. We omit the detailed verification of this fact for synchronous transitions as it can easily be obtained from the proof of Lemma 34. □

Synchronous Bisimulations

What about synchronous bisimilarity? It cannot coincide with $\mathcal{T}_{max}$, because as strong synchronous bisimilarity, it cannot identify $\mathsf{0}$ with fw_{xx}. But would it at least be preserved under parallel composition or renaming? Alas, not.

LEMMA 27 *Let* $\mathsf{P}_1 \approx \mathsf{P}_2$ *and* $\mathsf{Q}_1 \approx \mathsf{Q}_2$. *Then* $\mathsf{timer}^{t_1}(\mathsf{P}_1, \mathsf{Q}_1) \approx \mathsf{timer}^{t_2}(\mathsf{P}_2, \mathsf{Q}_2)$ *for all* $t_1, t_2 > 0$.

[1] This means that the timestepper ϕ can distinguish these two processes, despite the induced processes having essentially equivalent (name-passing) synchronisation trees [53, 120]. This suggests that once we allow time-passing to be observed, processes contain vital intensional information that is lost with the passage to conventional synchronisation trees. It appears straightforward to ameliorate this mismatch between the discriminating power of processes and synchronisation trees, for example by enriching the semantic trees with appropriate annotations that allow to identify "time-passing" τ-transitions. We do not pursue this matter further here.

PROOF: Consider the relation $\mathcal{R}$ given by

$$\mathsf{timer}^{t_1-i_1}(\mathsf{P}_1,\mathsf{Q}_1) \quad \mathcal{R} \quad \mathsf{timer}^{t_2-i_2}(\mathsf{P}_2,\mathsf{Q}_2))$$

where $0 \leq i_1 < t_1$ and $0 \leq i_2 < t_2$. We show that $\mathcal{R} \cup \approx$ is a bisimulation. To this end let

$$\mathsf{timer}^{t_1-i_1}(\mathsf{P}_1,\mathsf{Q}_1) \xrightarrow{l} \mathsf{R}. \tag{4.2}$$

By a trivial induction on the derivation of (4.2) there are only two cases: $l = \tau$ and $l = x(\vec{z})$. If the former, $\mathsf{R} = \phi(\mathsf{timer}^{t_1-i_1}(\mathsf{P}_1,\mathsf{Q}_1))$. We must now distinguish two subcases. The first is $i_1 = t_1 - 1$. Then $\mathsf{R} = \mathsf{Q}_1$ and

$$\mathsf{timer}^{t_2-i_2}(\mathsf{P}_2,\mathsf{Q}_2) \xrightarrow{\tau} \cdots \xrightarrow{\tau} \phi^{t_2-i_2}(\mathsf{timer}^{t_2-i_2}(\mathsf{P}_2,\mathsf{Q}_2)) = \mathsf{Q}_2 \approx \mathsf{Q}_1$$

is a matching transition sequence. On the other hand, if the second subcase, $0 \leq i < t_1 - 1$ obtains, then the empty transition sequence, starting and ending with $\mathsf{timer}^{t_2-i_2}(\mathsf{P}_2,\mathsf{Q}_2)$, matches (4.2). If $l = x(\vec{z})$ then $\mathsf{P}_1 = x(\vec{v}).\mathsf{P}'_1$, $\mathsf{R} = \mathsf{P}'_1\{\vec{v}/\vec{z}\}$ and easily $\mathsf{P}_1 \xrightarrow{x(\vec{z})} \mathsf{P}'_1\{\vec{v}/\vec{z}\}$. Using $\mathsf{P}_1 \approx \mathsf{P}_2$, we know a sequence

$$\mathsf{P}_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} \mathsf{P}''_2 \xrightarrow{x(\vec{z})} \mathsf{P}'''_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} \mathsf{P}''''_2 \approx \mathsf{P}'_1\{\vec{v}/\vec{z}\}.$$

must exist. But, as $\mathsf{P}_2 = x(\vec{w}).\mathsf{P}'_2$, P_2 does not have active timers, hence $\mathsf{P}_2 = \mathsf{P}''_2$. Clearly, then

$$\mathsf{timer}^{t_2-i_2}(\mathsf{P}_2,\mathsf{Q}_2) \xrightarrow{x(\vec{z})} \mathsf{P}'''_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} \mathsf{P}''''_2 \approx \mathsf{P}'_1\{\vec{v}/\vec{z}\},$$

matching (4.2), as required. □

Now we can state two crucial facts.

LEMMA 28 *1. ϕ does not preserve bisimilarity.*

2. $\sim \subset \approx$.

PROOF: Consider $\mathsf{P}^t = \mathsf{timer}^t(x.\overline{y}, 0)$. Then, by Lemma 27, $\mathsf{P}^3 \approx \mathsf{P}^1$, but: $\phi(\mathsf{P}^3) = \mathsf{P}^2 \not\approx \phi(\mathsf{P}^1) = 0$. This establishes (1). For (2), clearly $\sim$ is a bisimulation. In addition, $\mathsf{timer}^1(x.\overline{y},\overline{z}) \not\sim \mathsf{timer}^2(x.\overline{y},\overline{z})$, but the two terms are equated by $\approx$, as shown in Lemma 27. □

LEMMA 29 *1. If $\mathsf{P} \approx \mathsf{Q}$ then $(\nu x)\mathsf{P} \approx (\nu x)\mathsf{Q}$.*

2. $\approx$ is not closed under renaming or timestepping. Consequently, it is also not closed under parallel composition or input.

PROOF: Similar to the proof of Lemma 26, noting that Example 2 demonstrates failure of congruence under parallel composition for $\approx$, too. □

A Weak Expressivity Result

One key question that we would like to have a good answer to is whether π_t can be encoded into π_a in a convincing way. Unfortunately, we are only able to give a very partial answer.

DEFINITION 38 A function $[\![\ ]\!]$ from π_t into π is *complete*, if $[\![\mathsf{P}]\!] \approx [\![\mathsf{Q}]\!]$ implies $\mathsf{P} \approx \mathsf{Q}$. Conversely, if $\mathsf{P} \approx \mathsf{Q}$ implies $[\![\mathsf{P}]\!] \approx [\![\mathsf{Q}]\!]$, then $[\![\ ]\!]$ is *sound*. If $\mathsf{P} \approx \mathsf{Q}$ implies $[\![\mathsf{P}]\!] \approx [\![\mathsf{Q}]\!]$ for all those P and Q in π_t that are also processes in π, then $[\![\ ]\!]$ is *sound on* π. Clearly, the former form of soundness implies the latter. $[\![\ ]\!]$ is *compositional* if $[\![\mathsf{P}|\mathsf{Q}]\!] \approx [\![\mathsf{P}]\!]|[\![\mathsf{Q}]\!]$.

PROPOSITION 8 *There is no compositional encoding of the π-calculus with timers into the π-calculus without timers that is also sound on π and complete.*

PROOF: Define $\mathsf{P}^t = \mathsf{timer}^t(y.\overline{z}, 0)$ and $\mathsf{Q} = x$. Then clearly, $\mathsf{P}^1|\mathsf{Q} \not\approx \mathsf{P}^2|\mathsf{Q}$ and $\mathsf{P}^1 \approx \mathsf{P}^2$ (see the Example 2). But

$$\begin{array}{lcll} \mathsf{P}^1 \approx \mathsf{P}^2 & \Rightarrow & [\![\mathsf{P}^1]\!] \approx [\![\mathsf{P}^2]\!] & \text{by soundness on } \pi \\ & \Rightarrow & [\![\mathsf{P}^1]\!] \,|\, [\![\mathsf{Q}]\!] \approx [\![\mathsf{P}^2]\!] \,|\, [\![\mathsf{Q}]\!] & \text{as } \approx \text{ is a congruence} \\ & \Rightarrow & [\![\mathsf{P}^1 \,|\, \mathsf{Q}]\!] \approx [\![\mathsf{P}^2 \,|\, \mathsf{Q}]\!] & \text{by compositionality} \\ & \Rightarrow & \mathsf{P}^1 \,|\, \mathsf{Q} \approx \mathsf{P}^2 \,|\, \mathsf{Q} & \text{by completeness} \end{array}$$

But in the proof of Lemma 29 we showed that $\mathsf{P}^1 \,|\, \mathsf{Q} \not\approx \mathsf{P}^2 \,|\, \mathsf{Q}$. □

We believe that Proposition 8 is not a mere curiosity. On the contrary: we believe that there cannot be a reasonable encoding of π_t into untimed π-calculi, as long as one assumes reasonable equivalences on source and target calculi. Unfortunately, the state of the art in expressiveness theory does not provide tools to even state, let alone prove or disprove, such a conjecture with precision, because the various uses of "reasonable" have yet to solidify.

Asynchronous Transitions and Bisimulations

The failure of the various synchronous bisimilarities to equate fw_{xx} with $\mathsf{0}$ lead Honda and Tokoro to propose asynchronous transitions [57] which model asynchronous observers. Since $\mathcal{T}_{max}$ does equate fw_{xx} and $\mathsf{0}$, asynchronous bisimilarity may be a suitable candidate for our purposes. Unfortunately, the straightforward adaptation of Honda's and Tokoro's techniques to our setting does not work either, because the (PAR) rule

$$(\text{PAR}) \quad \frac{\mathsf{P} \xrightarrow{l}_a \mathsf{P}' \qquad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \,|\, \mathsf{Q} \xrightarrow{l}_a \mathsf{P}' \,|\, \phi(\mathsf{Q})}$$

does not connect asynchrony well with time passing. To see what goes wrong consider what it means to asynchronously observe a process. Observing a process

$$\text{(Out)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{0}}$$

$$\text{(In}_a\text{)}\quad \frac{}{\mathsf{0} \xrightarrow{x(\vec{z})}_a \overline{x}\langle\vec{z}\rangle}$$

$$\text{(Par)}\quad \frac{\mathsf{P} \xrightarrow{l}_a \mathsf{P}' \quad l \neq x(\vec{z}) \quad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{l}_a \mathsf{P}' \mid \phi(\mathsf{Q})}$$

$$\text{(Par}_a\text{)}\quad \frac{\mathsf{P} \xrightarrow{x(\vec{z})}_a \mathsf{P}'}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{x(\vec{z})}_a \mathsf{P}' \mid \mathsf{Q}}$$

$$\text{(Com)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid x(\vec{v}).\mathsf{Q} \xrightarrow{\tau}_a \mathsf{Q}\{\vec{y}/\vec{v}\}}$$

$$\text{(Rep)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid !x(\vec{v}).\mathsf{Q} \xrightarrow{\tau}_a \mathsf{Q}\{\vec{y}/\vec{v}\} \mid !x(\vec{v}).\mathsf{Q}}$$

$$\text{(TimeIn)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R}) \xrightarrow{\tau}_a \mathsf{Q}\{\vec{y}/\vec{v}\}}$$

$$\text{(Res)}\quad \frac{\mathsf{P} \xrightarrow{l}_a \mathsf{Q} \quad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)}{(\nu x)\mathsf{P} \xrightarrow{l}_a (\nu x)\mathsf{Q}}$$

$$\text{(Open)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q} \quad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle}_a \mathsf{Q}}$$

$$\text{(Idle)}\quad \frac{}{\mathsf{P} \xrightarrow{\tau}_a \phi(\mathsf{P})}$$

$$\text{(Cong)}\quad \frac{\mathsf{P} \equiv \mathsf{P} \quad \mathsf{P}' \xrightarrow{l}_a \mathsf{Q}' \quad \mathsf{Q}' \equiv \mathsf{Q}}{\mathsf{P} \xrightarrow{l}_a \mathsf{Q}}$$

Figure 4.4: An alternative account of asynchronous transitions for the asynchronous timed π-calculus.

sending a message, means interacting with it, consuming one unit of time. The (Par) rule ensures that this timestep permeates throughout all active processes. On the other hand, if we emit an output to the process under observation, so as to be able to infer that it may be inputting, we cannot, in our model, know when this output particle migrates. Hence the observation $\xrightarrow{x(\vec{v})}_a$ should not be associated with a time step. So (Par) above works incorrectly. We propose to split (Par) in two:

$$\text{(Par)}\frac{\mathsf{P} \xrightarrow{l}_a \mathsf{P}' \quad l \neq x(\vec{v}) \quad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{l}_a \mathsf{P}' \mid \phi(\mathsf{Q})} \qquad \text{(Par}_a\text{)}\frac{\mathsf{P} \xrightarrow{x(\vec{y})}_a \mathsf{P}'}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{x(\vec{y})}_a \mathsf{P}' \mid \mathsf{Q}}$$

Figure 4.4 presents the full inductive definition of $\rightarrow_a$. We continue with some basic facts about asynchronous transitions.

Lemma 30 *1.* $\mathsf{P} \xrightarrow{x(\vec{y})}_a \mathsf{Q}$ *if and only if* $\mathsf{Q} \equiv \overline{x}\langle\vec{y}\rangle \mid \mathsf{P}$.

2. $\mathsf{P} \stackrel{\tau}{\rightarrow}_a \mathsf{Q}$ *iff* $\mathsf{P} \rightarrow \mathsf{Q}$ *iff* $\mathsf{P} \stackrel{\tau}{\rightarrow}_s \mathsf{Q}$.

PROOF: By straightforward inductions. □

Alas $\sim_a$ is not perfect either, as the next lemma shows.

LEMMA 31 1. *If* $\mathsf{P} \sim_a \mathsf{Q}$ *then* $(\nu x)\mathsf{P} \sim_a (\nu x)\mathsf{Q}$.

2. *If* $\mathsf{P} \sim_a \mathsf{Q}$ *then* $\mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{P}) \sim_a \mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{Q})$.

3. $\sim_a$ *is not closed under renaming or timestepping. Consequently, it is also not closed under parallel composition or input.*

PROOF: Similar to the proof of Lemma 26, using Lemma 29.1 to demonstrate failure of congruence under parallel composition. □

The Largest Strong (A)Synchronous Bisimulation Congruences Contained in Strong (A)Synchronous Bisimilarity

Although $\sim_a$ is not yet the right equivalence to characterise $\mathcal{T}_{max}$, it is a good place to start from. We must ensure two things: ensure closure under timestepping and under renaming. We begin with the former.

DEFINITION 39 A *strong ϕ-bisimulation* is a strong asynchronous bisimulation that is also time-closed. We denote the largest strong ϕ-bisimulation by $\sim'_a$.

The existence of $\sim'_a$ is justified by the following simple lemma.

LEMMA 32 *Strong ϕ-bisimulations are closed under arbitrary unions. $\equiv$ and* id *are examples of strong ϕ-bisimulations.*

The next lemma shows that $\sim'_a$ is closer to being a congruence than the previous candidates.

LEMMA 33 1. *If* $\mathsf{P} \sim'_a \mathsf{Q}$ *then* $(\nu x)\mathsf{P} \sim'_a (\nu x)\mathsf{Q}$.

2. *If* $\mathsf{P} \sim'_a \mathsf{Q}$ *then* $\mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{P}) \sim'_a \mathsf{timer}^t(x(\vec{v}).\mathsf{R}, \mathsf{Q})$.

3. *If* $\mathsf{P} \sim'_a \mathsf{Q}$ *then* $\mathsf{P}|\mathsf{R} \sim'_a \mathsf{Q}|\mathsf{R}$.

PROOF: (1) and (2) are standard and (3) follows from time-closure. □

Unfortunately, we are not quite there yet, as the following lemma shows.

LEMMA 34 *Assume x, y, a, b are fresh and distinct names. Define*

$$\begin{aligned} \mathsf{P} &= (\nu a)(\overline{x}\langle a\rangle \,|\, !y(v).\overline{v}) \\ \mathsf{Q} &= (\nu a)(\overline{x}\langle a\rangle \,|\, !y(v).\overline{v} \;\; |\, \mathsf{timer}^1(y(v).(\overline{v} \,|\, \mathsf{timer}^1(a.\overline{b}, 0)), 0)). \end{aligned}$$

Then $\mathsf{P} \sim'_a \mathsf{Q}$ *but not* $\mathsf{P}\{x/y\} \sim'_a \mathsf{Q}\{x/y\}$.

PROOF: We define $\mathcal{R}$ (up to $\equiv$) by

$$\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \;\; \mathcal{R} \;\; \begin{cases} \mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \\ \mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \\ (\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v} \;\mid \mathsf{timer}^1(a.\overline{b}, 0)) \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \end{cases}$$

where c_i is a name and $\vec{d_i}$ is a tuple of names. By (VC) we can assume all these names to be distinct from a. Then $\mathcal{R} \cup \mathsf{id}$ is a strong ϕ-bisimulation. As

$$\begin{aligned} \mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle &= \phi(\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle) \\ &\equiv \phi(\mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle) \\ &= \phi((\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v} \mid \mathsf{timer}^1(a.\overline{b}, 0)) \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle), \end{aligned}$$

$\mathcal{R}$ is time-closed. To see that $\mathcal{R}$ is a strong asynchronous bisimulation, note that $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$ has the following transitions.

- $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$. It is matched by
  - $\mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \phi(\mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle) = \mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$,
  - $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$ and
  - $(\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v} \;\mid \mathsf{timer}^1(a.\overline{b}, 0)) \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$
- $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\overline{x}\langle(\nu a)a\rangle}_a !y(v).\overline{v} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$, matched as follows.
  - $\mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\overline{x}\langle(\nu a)a\rangle}_a !y(v).\overline{v} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$,
  - $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\overline{x}\langle(\nu a)a\rangle}_a !y(v).\overline{v} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$, and
  - $(\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v} \;\mid \mathsf{timer}^1(a.\overline{b}, 0)) \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\overline{x}\langle(\nu a)a\rangle}_a !y(v).\overline{v} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle$.
- If $c_{i_0} = y$ then $\vec{d}_{i_0} = (e)$ for some name $e \neq a$ and $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \mathsf{P} \mid \overline{e} \mid \Pi_{i\neq i_0}\overline{c_i}\langle\vec{d_i}\rangle$ The matching transitions are
  - $\mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \mathsf{P} \mid \overline{e} \mid \Pi_{i\neq i_0}\overline{c_i}\langle\vec{d_i}\rangle$,
  - $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \mathsf{P} \mid \overline{e} \mid \Pi_{i\neq i_0}\overline{c_i}\langle\vec{d_i}\rangle$, and
  - $(\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v} \;\mid \mathsf{timer}^1(a.\overline{b}, 0)) \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \mathsf{P} \mid \overline{e} \mid \Pi_{i\neq i_0}\overline{c_i}\langle\vec{d_i}\rangle$.
- $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{c(\vec{d})}_a \mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \mid \overline{c}\langle\vec{d}\rangle$, where $a \notin \{c, \vec{d}\}$. This is matched by
  - $\mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{c(\vec{d})}_a \mathsf{Q} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \mid \overline{c}\langle\vec{d}\rangle$
  - $\mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{c(\vec{d})}_a \mathsf{P} \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \mid \overline{c}\langle\vec{d}\rangle$
  - $(\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v} \;\mid \mathsf{timer}^1(a.\overline{b}, 0)) \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{c(\vec{d})}_a (\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v} \;\mid \mathsf{timer}^1(a.\overline{b}, 0)) \mid \Pi\overline{c_i}\langle\vec{d_i}\rangle \mid \overline{c}\langle\vec{d}$

In all cases, the transition pairs do not leave $\mathcal{R} \cup \mathsf{id}$ (up to $\equiv$), as required. The transitions of the "other side" of $\mathcal{R}$ are similar. We just present two. The transitions

$$\begin{array}{lll} \mathsf{Q} \,|\, \Pi\overline{c_i}\langle\vec{d_i}\rangle & \xrightarrow{\tau}_a & (\nu a)(\overline{x}\langle a\rangle \,|\, \overline{d} \,|\, !y(v).\overline{y} \,|\, \mathsf{timer}^1(a.\overline{b}, 0)) \,|\, \Pi_{i\neq i_0}\overline{c_i}\langle\vec{d_i}\rangle \\ \mathsf{Q} \,|\, \Pi\overline{c_i}\langle\vec{d_i}\rangle & \xrightarrow{\tau}_a & (\nu a)(\overline{x}\langle a\rangle \,|\, \overline{d} \,|\, !y(v).\overline{y}) \,|\, \Pi_{i\neq i_0}\overline{c_i}\langle\vec{d_i}\rangle \,|\, \Pi_{i\neq i_0}\overline{c_i}\langle\vec{d_i}\rangle \end{array}$$

are both matched by

$$\mathsf{P} \,|\, \Pi\overline{c_i}\langle\vec{d_i}\rangle \xrightarrow{\tau}_a \mathsf{P} \,|\, \overline{e} \,|\, \Pi_{i\neq i_0}\overline{c_i}\langle\vec{d_i}\rangle$$

for some appropriate i_0.

Now consider $\mathsf{P}\{x/y\}$ and $\mathsf{Q}\{x/y\}$. Clearly

$$\begin{array}{lll} \mathsf{Q}\{x/y\} & \xrightarrow{\tau}_a & (\nu a)(!x(v).\overline{v} \;\,|\, \overline{a} \,|\, \mathsf{timer}^1(a.\overline{b}, 0)) \\ & \xrightarrow{\tau}_a & !x(v).\overline{v} \,|\, \overline{b} \\ & \xrightarrow{\overline{b}}_a & !x(v).\overline{v}. \end{array}$$

But $b \notin \mathsf{fn}(\mathsf{P}\{x/y\})$, so there cannot be a transition sequence

$$\mathsf{P}\{x/y\} \underbrace{\xrightarrow{\tau}_a \ldots \xrightarrow{\tau}_a}_{\geq 0} \xrightarrow{\overline{b}}_a \mathsf{P}'.$$

This prevents $\sim'_a$ from being closed under renaming. □

Of course failure of renaming-closure has ramifications for congruency.

LEMMA 35 *$\sim'_a$ is not closed under any of the three forms of input prefixing, i.e.* $\mathsf{P} \sim'_a \mathsf{Q}$ *does not imply* $x(\vec{v}).\mathsf{P} \sim'_a x(\vec{v}).\mathsf{Q}$ *or* $!x(\vec{v}).\mathsf{P} !x(\vec{v}). \sim'_a \mathsf{Q}$ *or* $\mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{R}) \sim'_a \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R})$.

PROOF: Assume $\mathsf{P} \sim'_a \mathsf{Q}$ and choose a fresh name x. Towards a contradiction, let $x(\vec{v}).\mathsf{P} \sim'_a x(\vec{v}).\mathsf{Q}$. By Lemma 33.3 then $x(\vec{v}).\mathsf{P} \,|\, \overline{x}\langle a\rangle \;\sim'_a\; x(\vec{v}).\mathsf{Q} \,|\, \overline{x}\langle a\rangle$. Now $x(\vec{v}).\mathsf{P} \,|\, \overline{x}\langle a\rangle \xrightarrow{\tau}_a \mathsf{P}\{a/v\}$. Clearly, $x(\vec{v}).\mathsf{Q}|\overline{x}\langle\vec{a}\rangle$ has only two transitions. The target of of first, $x(\vec{v}).\mathsf{Q} \,|\, \overline{x}\langle a\rangle \xrightarrow{\tau}_a x(\vec{v}).\mathsf{Q} \,|\, \overline{x}\langle a\rangle$, has a barb at x that $\mathsf{P}\{a/v\}$ lacks, so the only way $x(\vec{v}).\mathsf{P} \,|\, \overline{x}\langle a\rangle \xrightarrow{\tau}_a \mathsf{P}\{a/v\}$ can be matched is by $x(\vec{v}).\mathsf{Q} \,|\, \overline{x}\langle a\rangle \xrightarrow{\tau}_a \mathsf{Q}\{a/v\}$ with $\mathsf{P}\{a/v\} \;\mathcal{R}\; \mathsf{Q}\{a/v\}$. This violates Lemma 34. The other two forms of input are similar. □

On closer inspection it becomes clear that this failure of closure under substitution is robust in the sense that it is not a mere curiosity of $\sim'_a$: we could replace asynchronous transitions with synchronous ones, we might use (weak) bisimilarities instead of strong ones or drop the time-closure requirement. In all cases our counterexample remains valid. This justifies our related claims in Lemmas 29 and 33. In summary, we have the following results.

PROPOSITION 9 *None of $\approx, \approx', \sim, \sim'$ is closed under name substitution and hence under the three forms of input prefixing (here $\sim'$ is obtained from $\approx$ and $\approx'$ from $\approx$ in the same way as $\sim'_a$ from $\sim_a$, by requiring time-closure).*

PROOF: Similar to Lemmas 34 and 35. □

Another peculiarity of the example demonstrating the failure of closure under renaming is that it uses nested timers. While it would be premature to venture a conjecture, it appears difficult to come up with counterexamples that do not nest timers.

Timer-Bisimulation

We are almost there! All that remains for congruency is closure under renaming. The next definition enforces just that.

DEFINITION 40 A *timer-bisimulation* is a strong ϕ-bisimulation $\mathcal{R}$ such that $\mathsf{P}\,\mathcal{R}\,\mathsf{Q}$ implies $\mathsf{P}\{x/y\}\,\mathcal{R}\,\mathsf{Q}\{x/y\}$ for all $x, y \in \mathcal{N}$. Clearly $\equiv$ and id are timer-bisimulations. By $\sim_a^c$ we denote the largest timer-bisimulation, which is easily seen to exist.

PROPOSITION 10 *$\sim_a^c$ is the largest strong asynchronous bisimulation contained in $\sim_a$ that is also a congruence.*

PROOF: Closure under restriction and $\mathsf{timer}^t(\mathsf{R}, \cdot)$ is straightforward. Verifying closure under parallel composition is immediate by time-closure. Closure under inputs is similarly easy using renaming-closure. Since congruency straightforwardly implies time-closure and renaming closure, the maximality is a consequence of $\sim_a^c$'s coinductive definition. □

Most of the remainder of this chapter establishes that $\mathcal{T}_{max}$ and $\sim_a^c$ are identical.

4.6.2 $\sim_a^c = \mathcal{T}_{max}$

In the untimed asynchronous π-calculus, asynchronous bisimilarity soundly approximates the corresponding maximum theory, but does not characterise it. This section shows that having timers available simplifies things: timer bisimilarity coincides with $\mathcal{T}_{max}$. The main technical challenge is to show that for every sound theory $\mathcal{T}$: whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q}$ then there must be a transition $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q}'$, such that $\mathcal{T}_{max} \vdash \mathsf{P}' = \mathsf{Q}'$. Our actual argument will be relatively simple, but as it requires a veritable amount of syntactic manipulation, we sketch it first before delving into the details.

First a syntactic convention: we will often have to deal with transitions $\xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a$ where the detailed nature (beyond sorting) of $\vec{y}$ or $\vec{z}$ does not matter. By that we mean that we might use processes like $\overline{x}\langle yz\rangle$, but could have used $\overline{x}\langle zy\rangle$ $\overline{x}\langle xw\rangle$

or $(\nu a)\overline{x}\langle ba\rangle$ $(a \neq x)$ without affecting the results that interest us. For brevity we write $\xrightarrow{\overline{x}\langle\ldots\rangle}_a$ to indicate this situation and hope not to confuse the reader. Similarly, abbreviating, say, $\xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \xrightarrow{\overline{x}\langle(\nu\vec{a})\vec{b}\rangle}_a$ to $\xrightarrow{\overline{x}\langle\ldots\rangle}_a \xrightarrow{\overline{x}\langle\ldots\rangle}_a$ is not intended to imply that $\vec{y} = \vec{a}$ or $\vec{z} = \vec{b}$. Conversely, if we write an output in full, then that indicates that the details of bound names and objects may matter.

Key Steps in the Proof

We establish $\mathcal{T}_{max} = \sim_a^c$ by showing that $\mathcal{T}_{max}$ is a timer-bisimulation and that $\sim_a^c$ is a sound theory. The latter is straightforward. As mentioned, the difficult bit of the former is to show that whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{P}'$ then some Q' exists such that $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{a})\vec{b}\rangle}_a \mathsf{Q}'$ with $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$. We know from Proposition 2 and Lemma 7 that $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{P}'$, $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ implies the existence of a transition $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{a})\vec{b}\rangle}_a \mathsf{Q}'$ with $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$. Well-sortedness of all processes under consideration guarantees that the vectors $\vec{z}$ and $\vec{b}$ are of equal length, say $\vec{z} = (z_0 \ldots z_{n-1})$ and $\vec{b} = (b_0 \ldots b_{n-1})$. What we want to show is that $\overline{x}\langle(\nu\vec{y})\vec{z}\rangle = \overline{x}\langle(\nu\vec{a})\vec{b}\rangle$. We do this in three steps.

- First we verify that the locations of free and bound names coincide. This means that $z_i \in \{\vec{y}\}$ if and only if $y_i \in \{\vec{a}\}$ and we can speak of *free* and *bound indices*.
- Next we establish that free names coincide at free indices. This means: $z_i \notin \{\vec{y}\}$ implies $z_i = b_i$ and vice versa, whenever i is a free index.
- Finally we prove bound names to coincide: whenever $z_i \in \{\vec{y}\}$ then $z_i = c_i$ and vice versa.

In the light of the previous proofs in this chapter, it would be tempting to consider a context

$$C[\cdot] = [\cdot] \,|\, \mathsf{timer}^1(x(\vec{v}).(\overline{r}|\Pi_{v_i \in \vec{v}}\overline{v_i}\langle\ldots\rangle), \ldots)$$

and use Proposition 2 and Lemma 7 together with reduction-closure. Unfortunately this is too simplistic because such a context cannot distinguish an output of, say, $\langle st\rangle$ from $\langle ts\rangle$. The problem is that we map a non-commutative operation (observation of tuples of names) to a commutative one (observation of the parallel composition of outputs on names). The key idea to overcome this problem is to indicate a name's position in a tuple by the number of times it is being output. This works because sound theories cannot equate processes like $\overline{x}|\overline{x}$ with $\overline{x}|\overline{x}|\overline{x}|\overline{x}$. More generally, we use a rudimentary form of traces and show that whenever two processes are related by a sound π_t-theory then one of them can output on a channel n times in a row but not $n+1$ times if and only if the other can. This suggests to use a context like.

$$C[\cdot] = [\cdot] \,|\, \mathsf{timer}^1(x(\vec{v}).(\overline{r} \mid \Pi_{v_i \in \vec{v}}\Pi_{j=1}^{f(i)}\overline{v_i}\langle\ldots\rangle), \ldots).$$

The question is now: what is an appropriate choice of f? It cannot be the identity or other slow-growing functions. The problem is that after we infer a reduction step

$$C[\mathsf{P}] \to (\nu\vec{y})(\mathsf{P}' \mid \overline{r} \mid \Pi_{z_i \in \vec{z}} \Pi_{j=1}^{i} \overline{z_i}\langle ... \rangle)$$

the residual P' might also output additional z_i's as long as i is a free index, because the interaction might have caused time-outs which in turn may have launched new active output particles.

To tackle this issue, we prove that every process has an apriori upper bound on how many consecutive outputs it can send on a fixed channel. Just counting the appropriate activated outputs in a process in not enough, as parallel composition with other processes can increase the length of traces. For example, $\mathsf{timer}^1(y, \overline{x}|\overline{x})$ has only one trace of outputs on x: the empty one. Similarly, the maximal trace of consecutive outputs on x of $\overline{x}|\mathsf{timer}^2(y, \overline{x}|\overline{x})$ is of length 1. As soon as we run both processes in parallel we get a trace of length 5. But if we count all outputs not under a replication, we do get a suitable bound. Replication needs not be considered because any output under a replication is preceded by a τ-action, preventing it from counting as a uniform trace.

We must construct the function f such that no matter how much P' contributes to any trace we are interested in, as long as it is below a certain threshold, we can still compute indices from trace lengths. It is easy to construct such functions as we will demonstrate.

This allows to show that $\overline{x}\langle(\nu\vec{y})\vec{z}\rangle$ and $\overline{x}\langle(\nu\vec{c})\vec{d}\rangle$ coincide on free names. Looking at traces of bound names in the same way does not make sense because these names are restricted. So we observe them from the inside: we position a process under the restriction which will signal successful observation to the outside, if and only if the received bound names exhibit the pattern we seek to match. This is possible using a long sequential composition of inputs that essentially inverts the process described above by mapping commutatively composed numbers back to tuples.

But is a context like

$$C[\cdot] = [\cdot] \mid \mathsf{timer}^1(x(\vec{v}).(\overline{r} \mid \Pi_{v_i \in \vec{v}} \Pi_{j=1}^{f(i)} \overline{v_i}\langle ... \rangle) \mid \mathsf{OM}^d_{\vec{z}}, ...)$$

enough? ($\mathsf{OM}^d_{\vec{z}}$, called *output-muncher* tests for the appropriateness of the received bound names.) It seems not, because it only allows to conclude that $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q}'$ and $\mathcal{T}_{max} \vdash C'[\mathsf{P}'] = C'[\mathsf{Q}']$ for some non-trivial context $C'[\cdot]$, but not $\mathcal{T}_{max} \vdash \mathsf{P}' = \mathsf{Q}'$. Unfortunately, we could not establish a Decomposition Theorem that would allow to remove $C'[\cdot]$ from $\mathcal{T}_{max} \vdash C'[\mathsf{P}'] = C'[\mathsf{Q}']$. The problem is that $\overline{x}\langle(\nu\vec{y})\vec{z}\rangle$ is a scope-opening operation and removing $(\nu\vec{y})$ from a process potentially allows new forms of interference, that might not have been taken into account in $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$.

On close inspection, it becomes apparent that the whole construction works because $C[\cdot]$ contains certain carefully chosen processes; the presence or absence

of other agents is irrelevant. That means we can change $C[\cdot]$ such that (roughly) $C'[\cdot] = (\nu\vec{y})[\cdot] \,|\, \mathsf{R} \,|\, \mathsf{S})$ for some fixed process S and arbitrary R. Since we have no constraints on R, any test that might distinguish P' from Q' can be made guarded under $(\nu\vec{y})$. Hence taking off the restriction (and removing S) cannot produce fundamentally new testing contexts and we can justifiably conclude to $\mathcal{T}_{max} \vdash \mathsf{P}' = \mathsf{Q}'$, cf. Decomposition Theorem 20.

Uniform Traces and π_t-Theories

We begin with a notion of trace and relate it to sound theories.

DEFINITION 41 The predicate $\downarrow_x^n$ is defined as follows: $\mathsf{P} \downarrow_x^n$ if and only if

- $\mathsf{P} \underbrace{\xrightarrow{\overline{x}\langle\ldots\rangle}_a \cdots \xrightarrow{\overline{x}\langle\ldots\rangle}_a}_{n} \mathsf{Q}$, but Q has no transition $\mathsf{Q} \xrightarrow{\overline{x}\langle\ldots\rangle}_a \mathsf{R}$.
- Whenever $\mathsf{P} \underbrace{\xrightarrow{\overline{x}\langle\ldots\rangle}_a \cdots \xrightarrow{\overline{x}\langle\ldots\rangle}_a}_{m} \mathsf{Q}$, and Q has no transition $\mathsf{Q} \xrightarrow{\overline{x}\langle\ldots\rangle}_a \mathsf{R}$ then $m \leq n$.

We call $\downarrow_x^n$ a *uniform trace-barb* and sequences such as $\mathsf{P} \underbrace{\xrightarrow{\overline{x}\langle\ldots\rangle}_a \cdots \xrightarrow{\overline{x}\langle\ldots\rangle}_a}_{n} \mathsf{Q}$ *uniform traces*.

Of course not all processes have interesting uniform traces. The next definition allows a rough but useful classification according to what kind of traces a process may exhibit.

DEFINITION 42 Let x be a name. A process P is *x-irrelevant* if for all $n \geq 0$: $\phi^n(\mathsf{P}) \not\xrightarrow{\overline{x}\langle\ldots\rangle}_a$. If P is not x-irrelevant, it is *x-relevant*.

The function to be defined next maps processes to natural numbers in a way that will later allow to use induction when proving facts about uniform trace-barbs.

DEFINITION 43 The function $\mathsf{tl}(\mathsf{P})$ maps processes to the number of ticks left in all of P's timers, active in $\phi^n(\mathsf{P})$ for at least one n.

$$\mathsf{tl}(\mathsf{P}) = \begin{cases} t + \mathsf{tl}(\mathsf{R}) & \mathsf{P} = \mathsf{timer}^t(\mathsf{Q}, \mathsf{R}) \\ \mathsf{tl}(\mathsf{Q}) + \mathsf{tl}(\mathsf{R}) & \mathsf{P} = \mathsf{Q}|\mathsf{R} \\ \mathsf{tl}(\mathsf{Q}) & \mathsf{P} = (\nu x)\mathsf{Q} \\ 0 & \text{otherwise} \end{cases}$$

The next lemma provides a rudimentary normal form for processes, used for reasoning about uniform traces.

LEMMA 36 *For all processes* P *and all names* x *there are index sets* I *and* I_t *such that*

$$\mathsf{P} \equiv (\nu\vec{a})(\Pi_{i\in I}\overline{x}\langle ...\rangle \mid \Pi_{i\in I_t}\mathsf{timer}^{t_i}(\mathsf{Q}_i, \mathsf{R}_i) \mid \mathsf{S})$$

where each R_i *is* x*-relevant and* S *is* x*-irrelevant.*

PROOF: By straightforward induction on the structure of P. □

PROPOSITION 11 *For all processes* P *and all names* x *there is a unique* $n \geq 0$ *such that* $\mathsf{P}\downarrow_x^n$.

PROOF: If there is no transition $\mathsf{P} \xrightarrow{\overline{x}\langle ...\rangle}_a$ then clearly $\mathsf{P}\downarrow_x^0$ does the job. So assuming $\mathsf{P} \xrightarrow{\overline{x}\langle ...\rangle}_a$ we proceed by induction on $\mathsf{tl}(\mathsf{P})$. For the base case $\mathsf{tl}(\mathsf{P}) = 0$ we know from Lemma 36 that

$$\mathsf{P} \equiv (\nu\vec{a})(\Pi_{i=1}^k\overline{x}\langle ...\rangle \mid \mathsf{R})$$

where $x \notin \{\vec{a}\}$, R is x-irrelevant and does not contain active timers. Then clearly $\mathsf{P}\downarrow_x^k$ is the unique appropriate trace-barb. For the inductive step, assume $\mathsf{tl}(\mathsf{P}) > 0$. By Lemma 36 again

$$\mathsf{P} \equiv (\nu\vec{a})(\Pi_{i=1}^k\overline{x}\langle ...\rangle \mid \Pi_{i=1}^m\mathsf{timer}^{t_i}(\mathsf{Q}_i, \mathsf{R}_i) \mid \mathsf{S})$$

where $x \notin \{\vec{a}\}$, $k, m \geq 1$, S is x-irrelevant and each R_i is x-relevant. Now $\mathsf{P} \xrightarrow{\overline{x}\langle ...\rangle}_a \mathsf{P}'$ means

$$\mathsf{P}' \equiv (\nu\vec{b})(\Pi_{i=1}^{k-1}\overline{x}\langle ...\rangle \mid \Pi_{i=1}^m\phi(\mathsf{timer}^{t_i}(\mathsf{Q}_i, \mathsf{R}_i)) \mid \phi(\mathsf{S}))$$

where $\{\vec{b}\} \subseteq \{\vec{a}\}$. As $m \geq 1$, $\mathsf{tl}(\mathsf{P}') < \mathsf{tl}(\mathsf{P})$, so by (IH) there is a unique n' with $\mathsf{P}'\downarrow_x^{n'}$. Hence $\mathsf{P}\downarrow_x^{n'+1}$. For uniqueness, assume $\mathsf{P}\downarrow_x^{m+1}$ (by our assumptions, $\mathsf{P}\downarrow_x^0$ is impossible). Then

$$\mathsf{P} \xrightarrow{\overline{x}\langle ...\rangle}_a \mathsf{P}'' \underbrace{\xrightarrow{\overline{x}\langle ...\rangle}_a \cdots \xrightarrow{\overline{x}\langle ...\rangle}_a}_{m} \mathsf{P}''' \not\xrightarrow{\overline{x}\langle ...\rangle}_a,$$

i.e. $\mathsf{P}'''\downarrow_x^{m+1}$. But clearly $\mathsf{P}' \equiv \mathsf{P}''$, hence P' and P'' have exactly the same transitions, which means $m = n'$. □

DEFINITION 44 The *maximal output*, $\mathsf{mo}(\mathsf{P})$ of a process P gives an upper bound on the number of consecutive outputs a process can do.

$$\begin{aligned}
\mathsf{mo}(\mathsf{0}) &= 0 \\
\mathsf{mo}(\overline{x}\langle\vec{y}\rangle) &= 1 \\
\mathsf{mo}(\mathsf{P}|\mathsf{Q}) &= \mathsf{mo}(\mathsf{P}) + \mathsf{mo}(\mathsf{Q}) \\
\mathsf{mo}(\mathsf{timer}^t(\mathsf{P}, \mathsf{Q})) &= \mathsf{mo}(\mathsf{Q}) \\
\mathsf{mo}(x(\vec{v}).\mathsf{P}) &= 0 \\
\mathsf{mo}(!x(\vec{v}).\mathsf{P}) &= 0
\end{aligned}$$

The *no-timer-point* of P, written $\mathsf{ntp}(\mathsf{P})$, is defined below.

$$\mathsf{ntp}(\mathsf{P}) = \begin{cases} 0 & \phi(\mathsf{P}) = \mathsf{P} \\ 1 + \mathsf{ntp}(\phi(\mathsf{P})) & \phi(\mathsf{P}) \neq \mathsf{P} \end{cases}$$

Then $\mathsf{sat}_x(\mathsf{P})$, the *$x$-saturation* of P is given

$$\mathsf{sat}_x(\mathsf{P}) = n \qquad \text{if and only if} \qquad \phi^{\mathsf{ntp}(\mathsf{P})}(\mathsf{P})\downarrow_x^n .$$

The following lemma connects uniform traces with $\mathsf{sat}_x(\cdot)$ and $\mathsf{mo}(\cdot)$.

LEMMA 37 *1. Assume that* $\mathsf{P} \xrightarrow{\overline{x_1}\langle\ldots\rangle}_a \cdots \xrightarrow{\overline{x_n}\langle\ldots\rangle}_a \mathsf{Q}$ *and* R *is a subterm of* Q. *Then* $\mathsf{sat}_x(\mathsf{R}) \leq \mathsf{mo}(\mathsf{P})$ *for all names* x.

2. Assume $\mathsf{sat}_x(\mathsf{P}) = 0$ *then* $\mathsf{sat}_x(\mathsf{P} \mid \Pi_{i=1}^n \overline{x}\langle\ldots\rangle) = n$

3. If $\mathsf{P}|\mathsf{Q}\downarrow_x^n$ *and* $\mathsf{sat}_x(\mathsf{P}) < n$ *then* $\mathsf{Q}\downarrow_x^m$ *implies* $m > 0$.

PROOF: Immediate from the definitions. □

Constructing the Location Indicator

As alluded to, we will have to construct a function $f : \{0, \ldots, n-1\} \to \mathbb{N}$ that is not only invertible but stably so. This means that we can obtain i not only from $f(i)$ but even from $f(i) + n_0$ provided n_0 stays within reasonable bounds. There are many suitable functions but we will only construct one.

DEFINITION 45 Let $f : A \to \mathbb{N}$ and $g : \mathbb{N} \to A$ be partial functions and $n_0 \in \mathbb{N}$. We call g a *n_0-inverse to f* if for all k with $0 \leq k < n_0$:

$$g(k + f(a)) = a \qquad \text{for all } a \in A.$$

If such a g exists, f is an *n_0-location indicator*.

The next lemma shows that n_0-location indicators exist.

LEMMA 38 *Let $f : A \to \mathbb{N}$ be an injective function and $n_0 \in \mathbb{N}$. Define the function $F : \mathcal{P}_{\mathrm{fin}}(A) \to \mathbb{N}$ by*

$$S \mapsto (n_0 + 1)\Sigma_{s \in S} 2^{2f(s)+1}$$

and the partial function $G : \mathbb{N} \to \mathcal{P}_{\mathrm{fin}}(A)$ by

$$n \mapsto \begin{cases} S & \text{if } |n - F(S)| < n_0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Then the following hold.

1. *$F(S) > n_0$ for all appropriate S.*

2. *(a) $S \neq T$ implies $|F(S) - F(T)| \geq 2(n_0 + 1)$.*

   (b) G is a partial function.

   (c) G is an n_0-inverse to F.

3. *Let $S, T \subseteq \mathbb{N}$ be finite. $F(S) = F(T)$ iff $S = T$.*

4. *Let $S, T \subseteq \mathbb{N}$ be finite and $a, b < n_0$. Then $a + F(S) \leq b + F(T)$ if and only if $F(S) \leq F(T)$.*

5. *Let $S \subseteq \mathbb{N}$ be finite. Assume that $\vec{T} = \{T_1, ..., T_m\}$ and $\vec{U} = \{U_1, ..., U_n\}$ $(m \leq n)$ are two partitions of S such that $T_i, U_i \neq \emptyset$ for all appropriate i. Assume that $\psi : \{1, ..., m\} \to \{1, ..., n\}$ is an injective function and that $a_1, ..., a_m, b_1, ..., b_m \in \mathbb{N}$ such that*

   - *$a_i + F(T_i) \leq b_i + F(U_{\psi(i)})$ for all $i \in \{1, ..., m\}$.*
   - *$\Sigma_i a_i < n_0$ and $\Sigma_i b_i < n_0$.*

   Then $\vec{T} = \vec{U}$.

PROOF: First, (1) is immediate from the definition of F. For (2), please recall that $\Sigma_{i=0}^{n} x^i = \frac{1-x^{n+1}}{1-x}$ for all real numbers $x \neq 0$ [46]. Clearly, f induces a strict order on A: $a \sqsubset b$ if and only if $f(a) < f(b)$. Naturally, f is strictly monotonically increasing with respect to this ordering. Now let $S, T \in \mathcal{P}_{fin}(A)$ and w.l.o.g. $F(S) > F(T)$. This means that some $s^+ \in S$ must exist such that $t \sqsubset s^+$ for all $t \in T$. If $T = \emptyset$ then easily

$$|F(S) - F(T)| \geq (n_0 + 1)2^{2f(s^+)+1} \geq 2(n_0 + 1).$$

Otherwise, there must be $t_{max} = \mathsf{max}(T \setminus S)$ (where $\mathsf{max}(T \setminus S)$ is taken with respect to $\sqsubset$). Then

$$\begin{aligned}
|F(S) - F(T)| &= (n_0 + 1)(\Sigma_{a \in S \setminus T} 2^{2f(a)+1} - \Sigma_{a \in T \setminus S} 2^{2f(a)+1}) \\
&\geq (n_0 + 1)(2^{2f(s^+)+1} - \Sigma_{i=0}^{2f(t_{max})+1} 2^i) \\
&= (n_0 + 1)(2^{2f(s^+)+1} - 2^{2f(t_{max})+2} + 1) \\
&\geq (n_0 + 1)(2^{2f(t_{max})+3} - 2^{2f(t_{max})+2} + 1) \\
&= (n_0 + 1)2^{2f(t_{max})+3} \\
&\geq 8(n_0 + 1)
\end{aligned}$$

This establishes (2a). Next, assume that $|n - F(S)| < n_0$ and $|n - F(T)| < n_0$. Then by the triangle-inequality:

$$|F(S) - F(T)| \leq |n - F(S)| + |n - F(T)| < 2n_0.$$

By (2a) above, that means that $S = T$. Hence G is indeed a partial function. Finally, for (2c), choose an appropriate S and k with $0 \leq k < n_0$. Then

$$|(k + F(S)) - F(S)| = k < n_0$$

hence $G(k + F(S)) = S$.

(3) is immediate from (2c) while (4) follows from (1). For (5), assume that $x_0 = \mathsf{max}(U_{\psi(i)} \setminus T_i)$ existed. Then for some unique i_0: $x_0 \in T_{i_0} \setminus U_{\psi(i)}$ By construction this means that

$$\left.\begin{array}{l} y \in U_{\psi(i_0)} \\ y > x_0 \end{array}\right\} \Rightarrow y \in T_{i_0}$$

But then

$$\begin{aligned}
&(a_i + F(T_{i_0})) - (b_i + F(U_{\psi(i_0)})) \\
&= a_i - b_i + (n_0+1)\left(\sum_{\substack{x\in T_{i_0}\\ x<x_0}} 2^{2x+1} + 2^{2x_0+1} + \sum_{\substack{x\in T_{i_0}\\ x>x_0}} 2^{2x+1} - \sum_{\substack{x\in U_{\psi(i_0)}\\ x<x_0}} 2^{2x+1} - \sum_{\substack{x\in U_{\psi(i_0)}\\ x>x_0}} 2^{2x+1}\right) \\
&= a_i - b_i + (n_0+1)\left(\sum_{\substack{x\in T_{i_0}\setminus U_{\psi(i_0)}\\ x<x_0}} 2^{2x+1} + 2^{2x_0+1} - \sum_{\substack{x\in T_{i_0}\setminus U_{\psi(i_0)}\\ x<x_0}} 2^{2x+1}\right) \\
&\geq a_i - b_i + (n_0+1)\left(2^{2x_0+1} - \sum_{x=0}^{2x_0-1} 2^x\right) \\
&= a_i - b_i + (n_0+1)(2^{2x_0}+1) \\
&\geq a_i - b_i + 2n_0 + 2 \\
&\geq -n_0 + 2n_0 + 2 \\
&> 0
\end{aligned}$$

Hence $a_i + F(T_{i_0}) > b_i + F(U_{\psi(i_0)})$, in contradiction to one of the assumptions. Consequently $U_{\psi(i)} \subseteq T_i$ for all i. This allows to obtain $F(U_{\psi(i)}) \leq F(T_i)$ for all i. By (4) then also $b_i + F(U_{\psi(i)}) \leq a_i + F(T_i)$, hence $b_i + F(U_{\psi(i)}) = a_i + F(T_i)$. By 4 $F(U_{\psi(i)}) = F(T_i)$. Applying 3 gives $U_{\psi(i)} = T_i$ for all i. As $\vec{T}$ and $\vec{U}$ both partition S, and $\vec{U}$ does not contain $\emptyset$ as a partition, it must be the case that $\vec{T} = \vec{U}$. □

The role of n_0 in our definition of n_0-location indicators is to give an upper bound on the amount of noise the n_0-location indicator can tolerate without loosing the ability to be inverted.

Output Munching

The next step towards our characterisation theorem is to define OM^b_m, the output muncher. Given a string $m = \langle x_0 ... x_{n-1} \rangle$ of names, what $\mathsf{P}|\mathsf{OM}^b_m$ does is indicate, via b, if P is capable of outputting on the names in m and in that order.

DEFINITION 46 Let $m = \langle x_0 ... x_{n-1} \rangle$ be an n-tuple of names. Define

$$\mathsf{OM}^b_m = (\nu a)(x_0(\vec{v}_0).....x_{n-1}(\vec{v}_{n-1}).\overline{a} \,|\, \mathsf{timer}^{n+1}(a.\overline{b}, 0))$$

where a is fresh and $x_i \notin \{\vec{v_j}\}$ for all i, j. We call OM^b_m and *output muncher* with *munch* m and *signal* a. We write $\mathsf{P}\downarrow_m$ if $\mathsf{P} \xrightarrow{\overline{x}_0\langle ... \rangle}_a \cdots \xrightarrow{\overline{x}_{n-1}\langle ... \rangle}_a \mathsf{Q}$ for some Q.

The following lemma states two essential facts about output munching.

LEMMA 39 *1.* $\mathsf{P}\downarrow_s$ *iff* $\mathsf{P}|\mathsf{OM}^a_s \Downarrow_a$, *where* a *is fresh.*

2. Let $\mathcal{T}$ *be a sound theory and* $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$. *Then* $\mathsf{P}\downarrow^n_x$ *iff* $\mathsf{Q}\downarrow^n_x$.

PROOF: (1) is by definition and for (2) let $s = \langle \underbrace{x...x}_{n} \rangle$. Then

$$\begin{aligned}
\mathsf{P}\downarrow^n_x &\Rightarrow \mathsf{P}\downarrow_s \\
&\Rightarrow \exists \mathsf{P}'.\mathsf{P}|\mathsf{OM}^a_s \twoheadrightarrow \mathsf{P}'\downarrow_a \\
&\Rightarrow \exists \mathsf{Q}'.\mathsf{Q}|\mathsf{OM}^a_s \twoheadrightarrow \mathsf{Q}'\downarrow_a \\
&\Rightarrow \mathsf{Q}\downarrow_s \\
&\Rightarrow \mathsf{Q}\downarrow^m_x \qquad \text{for some } m \geq n.
\end{aligned}$$

Similarly we establish:

$$\mathsf{P}\downarrow^m_x \Rightarrow \mathsf{Q}\downarrow^k_x \qquad \text{for some } k \geq m.$$

Then Proposition 11 guarantees that $k = n$, hence $m = n$. □

The Main Lemma

We have now assembled all the tools to prove the main lemma that carries most of the workload for our characterisation theorem.

LEMMA 40 *Let* $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ *and* $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{P}'$. *Then there is a transition* $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q}'$ *such that* $\mathcal{T}_{\max} \vdash \mathsf{P}' = \mathsf{Q}'$.

PROOF: Choose an arbitrary process R and an integer $t > 0$. Define number $n_0 = 1 + \mathsf{mo}(\mathsf{P}) + \mathsf{mo}(\mathsf{Q}) + \mathsf{mo}(\mathsf{R})$. It is an upper bound on the length of certain output traces that $\mathsf{P}|\mathsf{R}$ and $\mathsf{Q}|\mathsf{R}$ or their descendants can do. Define the function f on natural numbers by $i \mapsto (n_0+1)2^{2i+1}$. For finite subsets S of $\mathbb{N}$ set $F(S) = \Sigma_{i \in S} f(i)$. For every name x and every tuple of names $\vec{y}$ let $\mathsf{indices}_{\vec{y}}(x) = \{i \,|\, y_i = x\}$. Assume that $\vec{y} = \langle y_0 ... y_{m-1} \rangle$ and $\vec{z} = \langle z_0 ... z_{n-1} \rangle$ for some $m, n \geq 0$. Define

$$s = \langle \underbrace{y_0, \ldots, y_0}_{F(\mathsf{indices}_{\vec{z}}(y_0))}, \underbrace{y_1, \ldots, y_1}_{F(\mathsf{indices}_{\vec{z}}(y_1))}, \ldots, \underbrace{y_{m-1}, \ldots, y_{m-1}}_{F(\mathsf{indices}_{\vec{z}}(y_{m-1}))} \rangle.$$

Now we are ready to construct the context $C[\cdot]$ that will allow to use reduction-closure to force the existence of the transition mentioned in the statement of the lemma. Choose two fresh and distinct names a, b. Recalling from Theorem 20 that $\mathsf{delay}^t(\mathsf{S})$ stands for processes of the form $(\nu a)\mathsf{timer}^t(a.0, \mathsf{S})$ $(a \notin \mathsf{fn}(\mathsf{S}))$, we define

$$C[\cdot] \;=\; [\cdot] \,|\, \mathsf{timer}^1(x(\vec{v}).(\mathsf{R} \,|\, \mathsf{delay}^t(\overline{a} \,|\, \Pi_{i=0}^{n-1}\Pi_{j=1}^{f(i)}\overline{v_i}\langle ...\rangle \,|\, \mathsf{OM}_s^b)), 0)$$

where $\vec{v}$ is chosen such that $\mathsf{fn}(\mathsf{OM}_s^b) \cap \{\vec{v}\} = \emptyset$. By our convention about the presentation of tuples and by well-sortedness of all extant processes, we can assume $\vec{v} = \langle v_0 ... v_{n-1}\rangle$. Then:

$$\begin{array}{rcl} C[\mathsf{P}] & \rightarrow & (\nu\vec{y})(\mathsf{P}' \,|\, \mathsf{R} \,|\, \mathsf{delay}^t(\overline{a} \,|\, \Pi_{i=0}^{n-1}\Pi_{j=1}^{f(i)}\overline{z_i}\langle ...\rangle \,|\, \mathsf{OM}_s^b)) \\ & \equiv & (\nu\vec{y})(\mathsf{P}' \,|\, \mathsf{R} \,|\, \mathsf{delay}^t(\overline{a} \,|\, \mathsf{P}_{\mathit{free}} \,|\, \mathsf{P}_{\mathit{bound}} \,|\, \mathsf{OM}_s^b)) \\ & \stackrel{def}{=} & \mathsf{P}'' \end{array}$$

where – under the assumption that S_{free} is the set of all appropriate indices of free names in $\vec{z}$ and S_{bound} is its complement: $S_{\mathit{free}} = \{i \,|\, z_i \notin \{\vec{y}\}\}$ and $S_{\mathit{bound}} = \{i \,|\, z_i \in \{\vec{y}\}\}$ – we abbreviate:

$$\mathsf{P}_{\mathit{free}} = \Pi_{i\in S_{\mathit{free}}}\Pi_{j=1}^{f(i)}\overline{z_i}\langle ...\rangle \qquad\qquad \mathsf{P}_{\mathit{bound}} = \Pi_{i\in S_{\mathit{bound}}}\Pi_{j=1}^{f(i)}\overline{z_i}\langle ...\rangle$$

By strong reduction-closure and Proposition 3 we can find a reduction step $C[\mathsf{Q}]{\rightarrow}\mathsf{Q}''$ such that $\mathcal{T} \vdash \mathsf{P}'' = \mathsf{Q}''$. As $\mathsf{P}''\Downarrow_a$, by Lemma 13.2 also $\mathsf{Q}''\Downarrow_a$. But there is only one way that is possible. Applying Lemma 8.5 again allows to infer the existence of a transition

$$\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{c})\vec{d}\rangle}_a \mathsf{Q}'$$

such that

$$\begin{array}{rcl} \mathsf{Q}'' & = & (\nu\vec{c})(\mathsf{Q}' \,|\, \mathsf{R} \,|\, \mathsf{delay}^t(\overline{a} \,|\, \Pi_{i=0}^{n-1}\Pi_{j=1}^{f(i)}\overline{d_i}\langle ...\rangle \,|\, \mathsf{OM}_s^b)) \\ & \equiv & (\nu\vec{c})(\mathsf{Q}' \,|\, \mathsf{R} \,|\, \mathsf{delay}^t(\overline{a} \,|\, \mathsf{Q}_{\mathit{free}} \,|\, \mathsf{Q}_{\mathit{bound}} \,|\, \mathsf{OM}_s^b)) \end{array}$$

where $T_{\mathit{free}} = \{i | z_i \notin \{\vec{y}\}\}$, $T_{\mathit{bound}} = \{i | z_i \in \{\vec{y}\}\}$, $\mathsf{Q}_{\mathit{free}} = \Pi_{i\in T_{\mathit{free}}}\Pi_{j=1}^{f(i)}\overline{d_i}\langle ...\rangle$ and $\mathsf{Q}_{\mathit{bound}} = \Pi_{i\in T_{\mathit{bound}}}\Pi_{j=1}^{f(i)}\overline{d_i}\langle ...\rangle$. Now clearly

$$\mathsf{P}'' \underbrace{\rightarrow \ldots \rightarrow}_{t} \mathsf{P}''' \stackrel{def}{=} \phi^t(\mathsf{P}') \,|\, \phi^t(\mathsf{R}) \,|\, \overline{a} \,|\, \mathsf{P}_{\mathit{free}} \,|\, \mathsf{P}_{\mathit{bound}} \,|\, \mathsf{OM}_s^b$$

By induction on t we prove the existence of a process Q''' such that $\mathcal{T} \vdash \mathsf{P}''' = \mathsf{Q}'''$,

$$\mathsf{Q}'' \underbrace{\rightarrow \ldots \rightarrow}_{t} \mathsf{Q}''' \stackrel{def}{=} \mathsf{Q}'''' \,|\, \overline{a} \,|\, \mathsf{Q}_{\mathit{free}} \,|\, \mathsf{Q}_{\mathit{bound}} \,|\, \mathsf{OM}_s^b \qquad \text{and} \qquad \mathsf{Q}' \,|\, \mathsf{R} \underbrace{\rightarrow \ldots \rightarrow}_{t} \mathsf{Q}''''.$$

The base case, $t = 1$ uses the fact that $\mathsf{P}'''\downarrow_a$, so also $\mathsf{Q}'''\downarrow_a$ (Proposition 2.2 again). But this is only possible if $\mathsf{Q}'|\mathsf{R} \rightarrow \mathsf{Q}''''$ and

$$\mathsf{Q}''' \equiv \mathsf{Q}'''' \,|\, \overline{a} \,|\, \mathsf{Q}_{\mathit{free}} \,|\, \mathsf{Q}_{\mathit{bound}} \,|\, \mathsf{OM}_s^b$$

Reasoning for inductive step step is very similar, starting from the (IH).

The next step is to demonstrate that $\overline{x}\langle(\nu\vec{y})\vec{z}\rangle = \overline{x}\langle(\nu\vec{c})\vec{d}\rangle$. This means we need to show for all $i = 0...n-1$:

- $z_i \in \{\vec{y}\}$ if and only if $d_i \in \{\vec{c}\}$,
- $z_i \notin \{\vec{y}\}$ implies $z_i = d_i$ and
- $z_i \in \{\vec{y}\}$ implies $z_i = d_i$.

Now choose $z_i \in \{\vec{z}\} \setminus \{\vec{y}\}$. First we establish the existence of an index j such that $z_i = d_j$ and $d_j \in \{\vec{d}\} \setminus \{\vec{c}\}$. Assume $\mathsf{P}''' \downarrow^{k_i}_{z_i}$. Then by construction

$$2(n_0+1) \leq F(\mathsf{indices}_{\vec{z}}(z_i)) \leq k_i < F(\mathsf{indices}_{\vec{z}}(z_i)) + n_0. \tag{4.3}$$

By Proposition 11 and Lemma 39.2: $\mathsf{Q}''' \downarrow^{k_i}_{z_i}$. Also, by construction of n_0, Lemma 37.2 and the definition of $\mathsf{sat}_{z_i}(\cdot)$: $\mathsf{sat}_{z_i}(\mathsf{Q}') < n_0$, $\mathsf{sat}_{z_i}(\mathsf{Q}_{bound}) = 0$, $\mathsf{sat}_{z_i}(\mathsf{OM}^b_s) = 0$ and $\mathsf{sat}_{z_i}(\overline{a}) = 0$. Consequently,

$$\mathsf{sat}_{z_i}(\overline{a} \,|\, (\nu\vec{c})(\mathsf{Q}'''' \,|\, \mathsf{Q}_{bound} \,|\, \mathsf{OM}^b_s)) \quad < \quad n_0 \tag{4.4}$$

by Lemma 37.1. But then $\mathsf{Q}'' \downarrow^{k_i}_{z_i}$, Lemma 7 and (4.4) imply

$$\mathsf{Q}_{free} \downarrow^{l_i}_{z_i} \qquad \Rightarrow \qquad l_i > 0.$$

Thus $z_i = d_j$ for some $d_j \in \{\vec{d}\} \setminus \{\vec{c}\}$. Next we show that

$$\mathsf{indices}_{\vec{z}}(z_i) = \mathsf{indices}_{\vec{d}}(d_j).$$

Since $\mathsf{Q}''' \downarrow^{k_i}_{z_i}$, it must also be the case that $\mathsf{Q}''' \downarrow^{k_i}_{d_j}$, where

$$2(n_0+1) \leq F(\mathsf{indices}_{\vec{d}}(d_j)) \leq k_i < F(\mathsf{indices}_{\vec{d}}(d_j)) + n_0 \tag{4.5}$$

Now (4.3), (4.5) and the Triangle-Inequality together yield:

$$\begin{aligned} &|F(\mathsf{indices}_{\vec{d}}(d_j)) - F(\mathsf{indices}_{\vec{z}}(z_i))| \\ &\qquad \leq \quad |F(\mathsf{indices}_{\vec{d}}(d_j)) - k_i| + |F(\mathsf{indices}_{\vec{z}}(z_i)) - k_i| \\ &\qquad < \quad 2n_0 \end{aligned}$$

But by Lemma 38 this guarantees $\mathsf{indices}_{\vec{z}}(z_i) = \mathsf{indices}_{\vec{d}}(d_j)$, as required.

Since $\mathsf{P}''' \Downarrow_b$, also $\mathsf{Q}''' \Downarrow_b$ (Lemma 13.2). By Lemma 39.1 then $\mathsf{Q}''''|\mathsf{Q}_{free}|\mathsf{Q}_{bound}\downarrow_s$. By construction, name in s is also in $\{\vec{y}\}$. But we have just established that $d_i \in \{\vec{c}\}$ iff $z_i \in \{\vec{y}\}$, so Q_{free} cannot contribute to $\downarrow_s$. Hence

$$\mathsf{Q}''''|\mathsf{Q}_{bound}\downarrow_s \,.$$

By definition of $\downarrow_s$ this means that we can find processes

$$(\mathsf{Q}^i, \mathsf{Q}^i_{bound})_{i\in\{0,\ldots,m\}} \qquad \text{such that} \qquad \mathsf{Q}'''' = \mathsf{Q}^0 \qquad \mathsf{Q}_{bound} = \mathsf{Q}^0_{bound}$$

and

$$\mathsf{Q}^i \mid \mathsf{Q}^i_{bound} \underbrace{\xrightarrow{\overline{y_i}\langle\ldots\rangle}_a \cdots \xrightarrow{\overline{y_i}\langle\ldots\rangle}_a}_{F(\mathsf{indices}_{\vec{z}}(y_i))} \mathsf{Q}^{i+1} \mid \mathsf{Q}^{i+1}_{bound}$$

for each appropriate i. By partial induction on $i \in \{0, ..., m-1\}$ we now construct a function

$$\psi_i : \{0, ..., i\} \to \{0, ..., m-1\}$$

such that

- ψ_i is injective, and
- for all $j \in \{0, ..., i\}$: $c_{\psi_i(j)} = y_j$ and $F(\mathsf{indices}_{\vec{z}}(y_i)) \leq F(\mathsf{indices}_{\vec{d}}(c_{\psi_i(j)})) + n_0$.

For the base case $i = 0$ we know that

$$\mathsf{Q}^0 | \mathsf{Q}^0_{bound} \downarrow^{l_0}_{y_0} \qquad \text{implies} \qquad F(\mathsf{indices}_{\vec{z}}(y_0)) \leq l_0.$$

Since $n_0 < l_0$ by definition, the relevant uniform trace cannot come from just Q^0, Q^0_{bound} must also contribute (Lemma 39.2). This means that some $c_j \in \{\vec{c}\}$ must exists with $c_j = y_0$. Set

$$\psi_0 = \{(0, j)\}.$$

By Lemma 37.3

$$l_0 \leq F(\mathsf{indices}_{\vec{d}}(c_{\psi_0(0)})) + n_0,$$

as required.

For the inductive step, assume $i \in \{1, ..., m-2\}$. We know that

$$\mathsf{Q}^{i+1} \mid \mathsf{Q}^{i+1}_{bound} \underbrace{\xrightarrow{\overline{y_{i+1}}\langle\ldots\rangle}_a \cdots \xrightarrow{\overline{y_{i+1}}\langle\ldots\rangle}_a}_{F(\mathsf{indices}_{\vec{z}}(y_{i+1}))} \mathsf{Q}^{i+2} \mid \mathsf{Q}^{i+2}_{bound}$$

Hence $\mathsf{Q}^{i+1} \mid \mathsf{Q}^{i+1}_{bound} \downarrow^{l_{i+1}}_{y_{i+1}}$. But, as above, that means

$$F(\mathsf{indices}_{\vec{z}}(y_{i+1})) \leq l_{i+1}.$$

As with the base case that means Q^{i+1}_{bound}, too, must contribute. Hence we can find $c_{j_0} \in \{\vec{c}\}$ with $c_{j_0} = y_{i+1}$. But $y_{i+1} \notin \{\psi_i(0), ..., \psi_i(i)\}$, for otherwise we could find $k < i+1$ with $\psi_i(k) = y_{i+1} = y_k$, which is impossible by definition of $\vec{y}$. Hence ψ_{i+1}, given by

$$j \mapsto \begin{cases} \psi_i(j) & j \in \{0, ..., i\} \\ c_{j_0} & j = i+1 \end{cases}$$

is injective, hence bijective, as $\{\vec{c}\}$ and $\{\vec{y}\}$ have the same cardinality. Using Lemma 37.3 once again, we obtain

$$l_{i+1} \leq F(\mathsf{indices}_{\vec{d}}(c_{\psi_{i+1}(i+1)})) + n_0.$$

This concludes the induction.

The fact that ψ_{m-1} is bijective and that $C_{\psi_{m-1}(i)=y_i}$ together imply $\{\vec{y}\} = \{\vec{c}\}$. We now show that $\vec{y} = \vec{c}$. Recall S_{bound}: it is partitioned by

- $(\mathsf{indices}_{\vec{z}}(y_i))_{y_i \in \{\vec{y}\}}$ and by
- $(\mathsf{indices}_{\vec{z}}(c_{\psi_{m-1}(i)}))_{y_i \in \{\vec{y}\}}$.

The first partition is by definition and the second because ψ_{m-1} permutes $\{\vec{y}\}$. Clearly each partition consists only of non-empty sets. This means we can apply Lemma 38.5 and conclude that both partitions coincide. But then $d_i \in \{\vec{c}\}$ implies $d_i = z_i$, as required.

We have now established that $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q}'$. All that is left to do is to verify $\mathcal{T}_{max} \vdash \mathsf{P}' = \mathsf{Q}'$.

We know that $\mathcal{T} \vdash \mathsf{P}'' = \mathsf{Q}''$. Recalling that

$$\begin{aligned} \mathsf{P}'' &= (\nu\vec{y})(\mathsf{P}' \mid \mathsf{R} \mid \mathsf{delay}^t(\overline{a} \mid \Pi_{i=0}^{n-1}\Pi_{j=1}^{f(i)}\overline{z_i}\langle ...\rangle \mid \mathsf{OM}_s^b)) \\ \mathsf{Q}'' &= (\nu\vec{c})(\mathsf{Q}' \mid \mathsf{R} \mid \mathsf{delay}^t(\overline{a} \mid \Pi_{i=0}^{n-1}\Pi_{j=1}^{f(i)}\overline{d_i}\langle ...\rangle \mid \mathsf{OM}_s^b)) \end{aligned}$$

and that t and R were arbitrary, we can apply the Decomposition Theorem 19 to conclude that $\mathcal{T}_{max} \vdash \mathsf{P}' = \mathsf{Q}'$. □

The following immediate consequence will be useful later.

COROLLARY 1 *If* $\mathcal{T} \vdash (\nu\vec{y})(\mathsf{P}'|\overline{x}\langle\vec{z}\rangle) = \mathsf{Q}$, *then* $\mathsf{Q} \equiv (\nu\vec{y})(\mathsf{Q}'|\overline{x}\langle\vec{z}\rangle)$ *and* $\mathcal{T}_{\max} \vdash \phi(\mathsf{P}') = \phi(\mathsf{Q}')$.

PROOF: Easy from Lemma 40. □

Having done all the hard work, we can now reap the payoff.

THEOREM 25 *(Labelled Characterisation of* $\mathcal{T}_{\max}$*)* $\mathcal{T}_{\max} = \sim_a^c$.

PROOF: We start by showing that $\mathcal{T}_{max}$ is a strong asynchronous bisimulation. For this purpose, assume that $\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}$. If $\mathsf{P} \xrightarrow{x(\vec{y})}_a \mathsf{P}'$, then by Lemma 30.1 $\mathsf{P}' \equiv \overline{x}\langle\vec{y}\rangle \mid \mathsf{P}$ and $\mathsf{Q} \xrightarrow{x(\vec{y})}_a \overline{x}\langle\vec{y}\rangle \mid \mathsf{Q}$. By congruency: $\mathcal{T}_{max} \vdash \overline{x}\langle\vec{y}\rangle \mid \mathsf{P} = \overline{x}\langle\vec{y}\rangle \mid \mathsf{Q}$. If $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{P}'$, then Lemma 40 guarantee an appropriate transition $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q}'$ with $\mathcal{T}_{max} \vdash \mathsf{P}' = \mathsf{Q}'$. If $\mathsf{P} \xrightarrow{\tau}_a \mathsf{P}'$ we use strong reduction-closure to construct an appropriate matching transition. Hence $\mathcal{T}_{max} \subseteq \sim_a^c$. By Proposition 5, $\mathcal{T}_{max}$ is time-closed and Proposition 4 ensures renaming-closure, so $\mathcal{T}_{max}$ is a timer-bisimulation and $\mathcal{T}_{max} \subseteq \sim_a^c$.

For the reverse implication, we show that $\sim_a^c$ is a sound theory. Clearly $\mathsf{0} \not\sim_a^c \overline{x}$, so $\sim_a^c$ is consistent. The coincidence of $\rightarrow$ and $\xrightarrow{\tau}_a$ (Lemma 30) together with $\sim_a^c$ being a strong asynchronous bisimulation and a π_t-congruence (Proposition 10) guarantee strong reduction-closure. As to the identification of all insensitive terms, clearly $\{(\mathsf{P},\mathsf{Q}) \mid \mathsf{P},\mathsf{Q} \text{ insensitive}\}$ is a strong asynchronous timer bisimulation, hence $\sim_a^c \subseteq \mathcal{T}_{max}$, as required. □

Before using the characterisation to derive some equalities, it might be instructive to reflect on the proof of the Main Lemma. Why can't it be adapted to show that reduction congruence coincides with asynchronous bisimilarity in the untimed asynchronous π-calculus? After all, the key trick in the proof, mapping tuples of names to multisets, but such that the number of a name's occurrences allows to determine its positions in the tuple, does not rely on timers. If we had $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{P}'$ we could define

$$C[\cdot] = [\cdot] \mid x(\vec{v}).(\Pi_i \Pi_{j=1}^{f(i)} \overline{v_i}\langle ...\rangle \mid ...).$$

Then

$$C[\mathsf{P}] \rightarrow (\nu\vec{y})(\Pi_i \Pi_{j=1}^{f(i)} \overline{z_i}\langle ...\rangle \mid ...) \stackrel{def}{=} \mathsf{P}''$$

and reduction-closure would ensure a reduction sequence

$$C[\mathsf{P}] \twoheadrightarrow \mathsf{Q}'' \qquad \text{and} \qquad \mathcal{T} \vdash \mathsf{P}'' = \mathsf{Q}''.$$

We could show that P'' and Q'' had the same *weak* uniform trace barbs. But unlike in π_t, this does not imply $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q}'$, because Q could contain *forwarders*, which add weak barbs: if $\mathsf{R} \downarrow_x$ then $\mathsf{R} \mid \mathsf{fw}_{xy} \Downarrow_y$. This means that we cannot determine the identity of names exported by Q, only their pattern of coincidence. In π_t the situation is different: timers can ensure that no forwarding takes place. This is possible because forwarding takes time.

Another issue is that the asynchronous transition system $\xrightarrow{l}_a$, used for the Characterisation Theorem, splits (Par) to ensure broadcasting the flow of time only for τ-transitions and output actions. This makes sense for asynchronous calculi. Nevertheless, nothing in our development compellingly suggests that one cannot characterise $\mathcal{T}_{max}$ using an alternative asynchronous transition system not splitting (Par) but rather broadcasting time-passing for every action including inputs. We confess to having no idea if that is possible or not.

Some Examples

We now look at a few examples of using the Characterisation Theorem. Because $\mathcal{T}_{max}$ is such a fine equivalence, our examples will not be very complicated.

EXAMPLE 3 The identity forwarder fw_{aa} and $\mathsf{0}$ are strongly reduction congruent. To see this, define $\mathcal{R}$ up to $\equiv$ by

$$\mathsf{fw}_{xx} \,|\, \Pi_i \overline{y_i}\langle \vec{z_i}\rangle \;\; \mathcal{R} \;\; \Pi_i \overline{y_i}\langle \vec{z_i}\rangle \qquad \text{whenever } \{y_i, \vec{z_i}\} \subseteq \mathcal{N}.$$

Obviously $\mathcal{R}$ is time- and renaming closed. Since all occurring processes are timer-free, idle transitions can trivially be matched. The only vaguely interesting transition

$$\mathsf{fw}_{xx} \,|\, \overline{x}\langle \vec{a}\rangle \,|\, \Pi_i \overline{y_i}\langle \vec{z_i}\rangle \xrightarrow{\tau}_a \mathsf{fw}_{xx} \,|\, \overline{x}\langle \vec{a}\rangle \,|\, \Pi_i \overline{y_i}\langle \vec{z_i}\rangle$$

is clearly matched by the idle transition

$$\overline{x}\langle \vec{a}\rangle \,|\, \Pi_i \overline{y_i}\langle \vec{z_i}\rangle \xrightarrow{\tau}_a \overline{x}\langle \vec{a}\rangle \,|\, \Pi_i \overline{y_i}\langle \vec{z_i}\rangle.$$

The next example shows that we only really need timers of the form $\mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{Q})$. All other timers can be constructed from this basic form.

EXAMPLE 4 Define

$$\mathsf{T}_1 = \mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \qquad \mathsf{T}_{t+1} = \mathsf{timer}^{t+1}(x(\vec{v}).\mathsf{P}, \mathsf{T}_t)$$

Then

$$\mathcal{T}_{max} \vdash \mathsf{T}_t = \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q})$$

for all $t > 0$. A relation that witnesses this equation given by

$$\mathsf{T}_t \,|\, \Pi_{i=1}^n \overline{x_i}\langle \vec{y_i}\rangle \;\; \mathcal{R} \;\; \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \,|\, \Pi_{i=1}^n \overline{x_i}\langle \vec{y_i}\rangle.$$

It is easy to see that $\mathcal{R}$ is a timer-bisimulation, closed under renaming, as required.

EXAMPLE 5 In the asynchronous π-calculus, the asynchronous bisimilarity soundly approximates the maximum sound theory but does not characterise it: for example $\overline{x}\langle z\rangle$ and $\overline{x}\langle z\rangle|\mathsf{fw}_{yz}$ are equated by the latter relation but not by the former [61]. In π_t, these processes are of course also not related by $\sim_a^c$, because $\overline{x}\langle z\rangle|\mathsf{fw}_{yz}$ can never match the transition $\overline{x}\langle y\rangle \xrightarrow{\overline{x}\langle y\rangle}_a \mathsf{0}$, but the Characterisation Theorem guarantees that $\mathcal{T}_{max} \not\vdash \overline{x}\langle y\rangle = \overline{x}\langle z\rangle|\mathsf{fw}_{yz}$.

EXAMPLE 6 Parallel composition commutes with delay operators.

$$\mathcal{T}_{max} \vdash \mathsf{delay}^t(\mathsf{P}|\mathsf{Q}) = \mathsf{delay}^t(\mathsf{P})|\mathsf{delay}^t(\mathsf{Q})$$

It is straightforward to see that $\mathcal{R} \cup \mathsf{id}$ is a timer bisimulation, where $\mathcal{R}$ is given by

$$\mathsf{delay}^t(\mathsf{P}|\mathsf{Q}) \;\; \mathcal{R} \;\; \mathsf{delay}^t(\mathsf{P}) \,|\, \mathsf{delay}^t(\mathsf{Q}).$$

EXAMPLE 7 $\mathcal{T} \vdash \overline{x}\langle \vec{y}\rangle \,|\, \overline{x}\langle \vec{y}\rangle = \mathsf{delay}^1(\overline{x}\langle \vec{y}\rangle) \,|\, \overline{x}\langle \vec{y}\rangle$ but not $\mathcal{T} \vdash \overline{x}\langle \vec{y}\rangle = \mathsf{delay}^1(\overline{x}\langle \vec{y}\rangle)$. We omit the easy proof of these statements.

This final example shows another way of how decompositions may not be possible.

A Remark on Locality

In §2.7.1, we have discussed locality, a restriction that prevents processes to bind input names. This also makes sense for π_t: we obtain π_t^{loc} by restricting π_t to local processes only. In the next chapters, π_t^{loc} will be important and questions about its properties become unavoidable. Fortunately, all results established so far for π_t also hold for π_t^{loc}. It is easy to see by looking over all our proofs that none used any input binding in a way that was inconsistent with locality.

4.7 Alternative Equivalences?

The various characterisations of the largest sound π_t-theory are pleasant because they illuminate this fundamental equivalence from several angles and testify to its canonicity. Nevertheless, the situation is not satisfactory: $\mathcal{T}_{max}$ is too discriminating. In some sense this is unavoidable: time passing cannot be hidden and timers are a sensitive tool for measuring it. So we are really paying the price for congruency. Often, however, we want or even need to be less demanding. In this section we sketch three techniques that might prove useful as starting points for further inquiries towards coarser equivalences.

4.7.1 Timer Coarsening

Our timers are very fine-grained in that they can detect the presence or absence of strong barbs. If you look at the proofs in Sections 4.4, 4.5 and 4.6, we often use contexts like

$$(\nu\vec{a})([\cdot] \mid \mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{Q}))$$

to spot if a process has a strong barb at x. Although we do not do this here (but see Example 4), it can be shown that just adding a capability for the detection of strong barbs to the asynchronous π-calculus suffices for encoding π_t. One key feature of strong-barb detecting contexts such as the above is that they only use timers that time-out in one reduction step. This suggests that we might get coarser equivalences if we restrict processes such that all timers are of the form

$$\mathsf{timer}^{nt}(x(\vec{v}).\mathsf{P}, \mathsf{Q}))$$

where $n > 0$ is some fixed integer. We obtain the original π_t as one limit case and the theory of untimed observation of π_t as the other. This restriction accords well with contemporary computers where the granularity of time-measurement is several order of magnitude above that of the duration of atomic computational steps of the CPU.

Unfortunately, this form of timer coarsening does not seam to lead to coarser equivalences. Consider the case $n = 5$: the context

$$C[\cdot] = (\nu\vec{a})(\tau.\tau.\tau.\tau.[\cdot] \mid \mathsf{timer}^5(x(\vec{v}).\mathsf{P}, \mathsf{Q}))$$

also allows to test processes for the existence of a strong barb at x. We could modify the definition of sound theories to require only closure under reduction contexts, to prohibit contexts such as $C[\cdot]$, or something even more involved. But is it a good idea to give up on congruency? Maybe it isn't, maybe it is in the presence of timers, at least partially. It would appear that solid answers can only be given after a more thorough exploration of the design space.

4.7.2 Clock-Drift

Our timers are precise, they do not exhibit clock-drift. This is why the process

$$(\nu xy)(\mathsf{timer}^7(x.\overline{a}, 0) \mid \mathsf{timer}^7(y.0, \overline{x}))$$

will be equated with 0 by all reasonable equivalences. In many implementations of timers, the absence of clock-drift cannot be guaranteed, due often to imperfections of the underlying physical clocks or because of the way timer interrupts are handled. In the interest of realistic models it may then be good to admit a degree of clock-drift. This can easily be done in our model. Essentially all we need to do is generalise the timestepper *function* ϕ to binary timestepper *relations* and modify two reduction rules.

$$(\textsc{Par}) \quad \frac{\mathsf{P} \to \mathsf{P}' \quad \mathsf{Q}' \in \phi(\mathsf{Q})}{\mathsf{P}|\mathsf{Q} \to \mathsf{P}'|\mathsf{Q}'} \qquad\qquad (\textsc{Idle}) \quad \frac{\mathsf{Q} \in \phi(\mathsf{P})}{\mathsf{P} \to \mathsf{Q}}$$

The labelled semantics is obtained accordingly.

This definition works in the sense that it can express arbitrary discrete clock-drift. But it is too liberal as some valid choices of ϕ permit reductions such as

$$\overline{x}\langle\vec{y}\rangle \to 0.$$

One can require timestepper relations to meet all manner of constraints to root out such pathological behaviour. We continue by listing some of them.

Definition 47 A binary relation ϕ on π_t-processes is a *simple timestepper* if it meets the following requirements.

- If P does not contain active timers, then $\phi(\mathsf{P}) = \{\mathsf{P}\}$.
- If $\mathsf{R} \in \phi(\mathsf{P}|\mathsf{Q})$ then $\mathsf{R} = \mathsf{P}'|\mathsf{Q}'$ where $\mathsf{P}' \in \phi(\mathsf{P})$ and $\mathsf{Q}' \in \phi(\mathsf{Q})$.
- If $\mathsf{Q} \in \phi((\nu x)\mathsf{P})$ then $\mathsf{Q} = (\nu x)\mathsf{P}'$ for some $\mathsf{P}' \in \phi(\mathsf{P})$.

- If $\mathsf{R} \in \phi(\mathsf{timer}^t(\mathsf{P},\mathsf{Q}))$ then $\mathsf{R} = \mathsf{timer}^{t'}(\mathsf{P},\mathsf{Q})$ or $\mathsf{R} = \mathsf{Q}$.

Clearly, if we restrict ϕ-relations to simple timesteppers, reductions such as $\overline{x}\langle\vec{y}\rangle \rightarrow \mathsf{0}$ are no longer possible. It is worthwhile to note that taking the identity relation as the timestepper yields a version of the asynchronous π-calculus, where we have two syntactically different, yet behaviourally indistinguishable input-prefixes, $x(\vec{v}).\mathsf{P}$ and $\mathsf{timer}^t(x(\vec{v}).\mathsf{P},\mathsf{Q})$. To exclude this limit case we can impose

$$\mathsf{timer}^t(\mathsf{P},\mathsf{Q}) \in \phi(\mathsf{timer}^{t+1}(\mathsf{P},\mathsf{Q})) \qquad \text{and} \qquad \mathsf{Q} \in \phi(\mathsf{timer}^1(\mathsf{P},\mathsf{Q})).$$

In addition, we might want clock-drift to be uniform in the sense of context independence: if

$$\mathsf{timer}^7(\mathsf{P},\mathsf{Q})|\mathsf{R}' \in \phi(\mathsf{timer}^9(\mathsf{P},\mathsf{Q})|\mathsf{R})$$

then also

$$\mathsf{timer}^7(\mathsf{P},\mathsf{Q})|\mathsf{S}' \in \phi(\mathsf{timer}^9(\mathsf{P},\mathsf{Q})|\mathsf{S}).$$

It seems reasonable to require clock-drift to be independent of the choice of continuations of the relevant timers. The following two statements show how to formalise this. For all reduction contexts $C_1[\cdot], C_2[\cdot]$, $t, t', \mathsf{P}_i, \mathsf{Q}_i$,

$$\begin{gathered} C_1'[\mathsf{timer}^{t'}(\mathsf{P}_1,\mathsf{Q}_1)] \in \phi(C_1[\mathsf{timer}^t(\mathsf{P}_1,\mathsf{Q}_1)] \\ \text{iff} \\ C_2'[\mathsf{timer}^{t'}(\mathsf{P}_2,\mathsf{Q}_2)] \in \phi(C_2[\mathsf{timer}^t(\mathsf{P}_2,\mathsf{Q}_2)]. \end{gathered}$$

Similarly, for all reduction contexts $C_1[\cdot], C_2[\cdot]$, $t, \mathsf{P}_i, \mathsf{Q}_i$,

$$\mathsf{Q}_1 \in \phi(C_1[\mathsf{timer}^t(\mathsf{P}_1,\mathsf{Q}_1)] \quad \text{iff} \quad \mathsf{Q}_2 \in \phi(C_2[\mathsf{timer}^t(\mathsf{P}_2,\mathsf{Q}_2)].$$

Interestingly, uniformity of clock-drift might not always be desirable in models. Since timers are often implemented via interrupts which can be disabled by other programs, the execution context of a program can influence clock-drift. A typical example would be when the OS temporarily blocks timer interrupts to ensure swift completion of some higher priority task.

Another type of uniformity one might require is independence of clock-drift from the timers' time-out time:

$$\mathsf{timer}^{t-\Delta}(\mathsf{P},\mathsf{Q}) \in \phi(\mathsf{timer}^t(\mathsf{P},\mathsf{Q})) \qquad \text{iff} \qquad \mathsf{timer}^{t'-\Delta}(\mathsf{P},\mathsf{Q}) \in \phi(\mathsf{timer}^{t'}(\mathsf{P},\mathsf{Q}))$$

for all $t, t' > 0$, $\Delta \in \mathbb{Z}$, and all $t - \Delta, t' - \Delta > 0$.

It is not clear that we have exhausted the uniformity conditions one might reasonably stipulate, or even that they are appropriately approximated by our suggestions above. As much in this text, we have intended to provide some reasonable starting points for further investigations.

Legitimisation

Being a simple timestepper is often not enough, even with the additional uniformity requirements sketched in the last section, because of lacking constraints on the evolution of clock-drift in the course of a computation. For example we have not prohibited

$$\mathsf{timer}^{37482}(P,Q) \in \phi(\mathsf{timer}^{1}(P,Q)) \qquad \text{and} \qquad \mathsf{timer}^{1}(P,Q) \in \phi(\mathsf{timer}^{12346}(P,Q)).$$

Given the discrepancy between clock speeds of modern CPUs and the granularity of timers, a sloppy implementation of the OS's interrupt mechanisms may lead to such clock-drift. Nevertheless, sometimes such behaviour may be unacceptable. For precise models, we need to be able to finely control the permitted clock-drift. This touches on two issues: how to measure clock-drift and how to exclude computations with inappropriately drifting clocks? The latter problem is easily solved with the following informally presented construction.

DEFINITION 48 Given a graph $G = (V, \rightarrow)$, a *legitimiser* is a unary predicate $\Box$ on finite paths in G such that $\Box$ is prefix closed, contains the empty string and is closed under concatenation of finite reduction paths. Legitimisers for labelled graphs are defined similarly. We will usually write $\Box(\mathsf{P} \twoheadrightarrow \mathsf{Q})$ for $(\mathsf{P} = \mathsf{P}_0, \mathsf{P}_1, \ldots, \mathsf{P}_n = \mathsf{Q}) \in \Box$, hoping that this will not confuse the reader.

For the graphs we are interested in, those induced by the reduction relation of process calculi, we will in general impose additional closure requirements, for example under $\equiv$ or injective renaming.

Given a process calculus and a legitimiser $\Box$, we obtain a $\Box$-equivalent of every construction in the original calculus by simply replacing references to traces $\mathsf{P} \twoheadrightarrow \mathsf{Q}$ with references to $\Box$-legitimised traces $\Box(\mathsf{P} \twoheadrightarrow \mathsf{Q})$.

EXAMPLE 8 Let $\Box$ be a legitimiser for the π_t-calculus. A π_t-theory $\mathcal{T}$ is $\Box$-reduction-closed if $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\Box(\mathsf{P} \twoheadrightarrow \mathsf{P}')$ implies the existence of a process Q' such that $\Box(\mathsf{Q} \twoheadrightarrow \mathsf{Q}')$ and $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$. A process P is $\Box$-insensitive if $\Box(\mathsf{P} \twoheadrightarrow \mathsf{Q})$ implies $\mathsf{an}(\mathsf{Q}) = \emptyset$.

If $C[\cdot]$ and $C'[\cdot]$ are reduction contexts, then $C[\cdot]$ *$\Box$-generically reduces to* $C'[\cdot]$, denoted $\Box(C[\cdot] \twoheadrightarrow C'[\cdot])$ if some legitimised reduction sequence $\Box(C[\mathsf{P}] \twoheadrightarrow C'[\mathsf{P}])$ exists for all P.

A π_t-theory is *$\Box$-sound* if it is consistent, $\Box$-reduction-closed and equates any two $\Box$-insensitive processes.

All equivalences we have touched upon in this text can be $\Box$-ified in this way. What kind of theorems hold for for $\Box$-equivalences depends on $\Box$ of course, but some general structure seems to be available: we conjecture that for many choices of $\Box$

and $\Box' \, \Box \subseteq \Box'$ implies $\mathcal{R}_{\Box'} \subseteq \mathcal{R}_{\Box}$, where $\mathcal{R}$ is one of the standard equivalences for π-calculi and $\mathcal{R}_{\Box}$ its $\Box$-ified version, ditto for $\mathcal{R}_{\Box'}$.

How does that relate to clock-drift? Clock-drift is usually specified relative to duration: "this clock does not drift more then 3 seconds per year!" As we measure time through the number of reduction steps in π_t, it appears natural to specify clock-drift along the lines of: "in traces of length n we allow at most 3 ticks clock-drift!" Assuming we know how to measure the clock-drift of reduction sequence we can then use appropriate legitimisers:

$$\Box_{\frac{3}{100}} = \{\mathsf{P} \twoheadrightarrow \mathsf{Q} \mid \frac{\mathsf{cd}(\mathsf{P} \twoheadrightarrow \mathsf{Q})}{\mathsf{length}(\mathsf{P} \twoheadrightarrow \mathsf{Q})} \leq \frac{3}{100}\}$$

All that remains is to explain how to define $\mathsf{cd}(\mathsf{P} \twoheadrightarrow \mathsf{Q})$, the *clock-drift* of $\mathsf{P} \twoheadrightarrow \mathsf{Q}$. Unfortunately, we know of no canonical way of doing this. Consider the trace

$$\mathsf{timer}^3(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^2(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{Q}.$$

We want $\mathsf{cd}(\mathsf{timer}^3(\mathsf{P},\mathsf{Q}) \twoheadrightarrow \mathsf{Q}) = 0$, but what about

$$\mathsf{timer}^3(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{Q}?$$

It is reasonable to say that $\mathsf{cd}(\mathsf{timer}^3(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q})) = 1$ because the clock makes one tick too many. Conversely, $\mathsf{cd}(\mathsf{timer}^1(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q})) = -1$. But does that mean that

$$\mathsf{cd}(\mathsf{timer}^3(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q})) = 0$$

because the clock's two missteps cancel each other out? Or is it better to add:

$$\mathsf{cd}(\mathsf{timer}^3(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q}) \rightarrow \mathsf{timer}^1(\mathsf{P},\mathsf{Q})) = 2?$$

And what about active timers running in parallel? Do we add clock-drift, as in

$$\mathsf{cd}(\mathsf{timer}^4(\mathsf{P},\mathsf{Q})|\mathsf{timer}^8(\mathsf{P}',\mathsf{Q}') \rightarrow \mathsf{timer}^2(\mathsf{P},\mathsf{Q})|\mathsf{timer}^1(\mathsf{P}',\mathsf{Q}')) = 1 + 6?$$

Or should we take the maximum

$$\mathsf{cd}(\mathsf{timer}^4(\mathsf{P},\mathsf{Q})|\mathsf{timer}^8(\mathsf{P}',\mathsf{Q}') \rightarrow \mathsf{timer}^2(\mathsf{P},\mathsf{Q})|\mathsf{timer}^1(\mathsf{P}',\mathsf{Q}')) = \mathsf{max}(1,6)?$$

The right choice may depend on the application, but once it is made, it induces a legitimiser which in turn determines equivalences.

4.7.3 Idling

As discussed in §4.3.1, our semantics is temporally asynchronous as we can always derive $\mathsf{P} \to \phi(\mathsf{P})$. Other models of timed computation make different choices with the opposite extreme being the requirement of maximal progress which allows idling only when no non-idling reduction steps are derivable. Using legitimisers we can easily describe such behaviour without having to modify our calculus.

DEFINITION 49 A reduction step $\mathsf{P} \to \mathsf{Q}$ is *idling* if each of its derivations contains an application of (IDLE). If, in addition, every reduction $\mathsf{P} \to \mathsf{R}$ is idling, we say $\mathsf{P} \to \mathsf{Q}$ is an *essential idling*. An idling that is not essential is *inessential*.

DEFINITION 50 Call a reduction sequence $\mathsf{P}_0 \to \cdots \to \mathsf{P}_n$ *maximally progressing* if each constituent $\mathsf{P}_i \to \mathsf{P}_{i+1}$ is either not idling or an essential idling. With the π_t-legitimiser

$$\Box_{mp} = \{\mathsf{P} \twoheadrightarrow \mathsf{Q} \mid \mathsf{P} \twoheadrightarrow \mathsf{Q} \text{ is maximally progressing}\}$$

we obtain the maximally progressing, timed, asynchronous π-calculus.

Analogously to what we did with clock-drift, we can also deal with degrees of idling.

DEFINITION 51 Let $0 \leq p \leq q$. Define $\Box_p$ as the largest π_t-legitimiser such that $\Box_p(\mathsf{P}_0 \to \mathsf{P}_1 \to \ldots \to \mathsf{P}_n)$ $(n > 0)$ implies

$$\frac{\text{number of inessential idles in } \mathsf{P}_0 \to \ldots \to \mathsf{P}_n \to \mathsf{P}_{n+1}}{n} \geq p,$$

Clearly, $\Box_{mp} = \Box_0$. A shortcoming of this definition is that for $0 \leq p < 1$, inessential idles cannot be initial steps in reduction sequences legitimised by $\Box_p$.

Removing (IDLE) Altogether

The (IDLE) rule was introduced to prevent the flow of time from ever stopping, as would otherwise be possible, through deadlocked processes. But if we could ensure that no valid process could ever deadlock, then (IDLE) would be superfluous.

DEFINITION 52 Denote by π_{t}^{-} the calculus obtained from π_{t} by removing the (IDLE) rule in the reduction and the transition semantics. The following function embeds

π_{t}^{-} into π_{t}.

$$\begin{aligned}
[\![x(\vec{y}).\mathsf{P}]\!] &= \Omega|x(\vec{y}).[\![\mathsf{P}]\!] \\
[\![!x(\vec{y}).\mathsf{P}]\!] &= \Omega|!x(\vec{y}).[\![\mathsf{P}]\!] \\
[\![\overline{x}\langle\vec{y}\rangle]\!] &= \Omega|\overline{x}\langle\vec{y}\rangle \\
[\![\mathsf{P}|\mathsf{Q}]\!] &= [\![\mathsf{P}]\!]|[\![\mathsf{Q}]\!] \\
[\![(\nu x)\mathsf{P}]\!] &= (\nu x)[\![\mathsf{P}]\!] \\
[\![\mathsf{0}]\!] &= \Omega \\
[\![\mathsf{timer}^t(x(\vec{v}).\mathsf{P},\mathsf{Q})]\!] &= \Omega|\mathsf{timer}^t(x(\vec{v}).[\![\mathsf{P}]\!],[\![\mathsf{Q}]\!])
\end{aligned}$$

Here Ω is a divergent process that does not admit observations of names: $\mathsf{fn}(\Omega) = \emptyset$, $\phi(\Omega) = \Omega$, $\Omega \xrightarrow{\pi} \mathsf{P}$ implies $\mathsf{P} \equiv \Omega, \pi = \tau$. For example, $(\nu x)(\overline{x}|!x.\overline{x})$ would be suitable.

We conjecture that this embedding is fully abstract with respect to most reasonable choices of equivalence.

4.8 Alternative Approaches

There are a large number of proposals in the literature for adding time to process calculi; to such an extent that it is very difficult to compare and contrast the often competing schemes. Matthew Hennessy [49].

Having explored the timed π-calculus in some detail, we briefly summarise alternative approaches. But first a word of warning: the proliferation of formal approaches to timed computation has not diminished since the introductory quote was written. A comprehensive unification is also still lacking. Consequently, the following overview of alternatives to π_t is not aiming at completeness. We merely wish to sketch some eclectically chosen alternatives, so as to allow the reader to better understand and position π_t. It is loosely structured by six questions that all relevant formalisms will have to address. The questions are of varying importance and hardly orthogonal, but seem well suited to bring out π_t's key features in comparison with its alternatives and predecessors. Here they are.

- What is the time domain?
- What is the underlying untimed computational formalism?
- What kind of semantics is used?
- How is the flow of time propagated?
- What constructs, if any, are used to be able to explicitly make the computation contingent upon temporal properties.

- What are the progress assumptions?

Time Domain?

There are really only three popular time domains: *discrete time*, *dense time* and *continuous time*, i.e. $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$. Of course one can envision other choices, for example partial orders that are not total. But that is likely to result in calculi with weird temporal properties, far removed from current intuitions about the relationship between time-passing and computation. Discrete time has been used for π_t because it is simple and models most faithfully what is going on in the kind of systems that are prevalent as computers today. These systems are all driven by a clocking mechanism and its periodicity is a lower bound on the duration of temporal events that are of relevance. But for hybrid systems, where discrete, digital systems interact with continuous, analog ones, having $\mathbb{R}$ as time domains seems to have advantages. With dense and continuous time, pathological behaviour is possible that does not affect discrete time models. One example are *Xeno* computations that make infinitely many steps in finite time, although it is generally possible to avoid such paradoxes with a careful setup of what counts as a computation.

Axiomatic treatments of timed computation are usually parametrised by a time domain. For example [64, 65] use a commutative monoid $(T, +, 0)$ such that the induced canonical preorder ($s \leq t$ iff for some u: $s+u = t$) is linear and $s+t = s+u$ implies $t = u$. It would have been possible to have parametrised our presentation of π_t in a similar fashion, but for simplicity we have chosen not to.

Underlying Untimed Computational Formalism?

There are many models of untimed computations. Decades of research have shown that despite the Church-Turing Thesis, they differ in their properties and expressiveness. All of them can be turned into models of timed computation, along the lines of our transformation of the asynchronous π-calculus into π_t, but not much is know about how this metamorphosis affects these expressivity differences. One of the triggers for the development of π_t was the assumption that these differences will be preserved by addition of timers. More concretely: we have assumed that the ability to create private channels dynamically was orthogonal to time passing and its manipulation by timers. If that was true, a timed π-calculus would most likely be more expressive than timed CCS. It seems reasonable to expect that many other properties of the various untimed models of computation are similarly reflected by their timed counterparts.

To see just how easy it is to take a model of untimed computation and turn it into a timed model, we give an example. Consider λ_t, a timed λ-calculus. Its terms

are

$$M ::= x \mid (\lambda x.M) \mid (MN) \mid \mathsf{timer}^t(\lambda x.M, N).$$

Its β-reduction has now got two defining rules

$$\begin{aligned} (\lambda x.M)N &\rightarrow M\{N/x\} \\ \mathsf{timer}^t(\lambda x.L, M)N &\rightarrow L\{N/x\} \end{aligned}$$

while the key contextual rules are

$$\begin{aligned} M \rightarrow M' &\Rightarrow (MN) \rightarrow (M'\phi(N)) \\ N \rightarrow N' &\Rightarrow (MN) \rightarrow (\phi(M)N'). \end{aligned}$$

Here $\phi(\mathsf{timer}^{t+1}(\lambda x.M, N)) = \mathsf{timer}^t(\lambda x.M, N)$ and $\phi(\mathsf{timer}^1(\lambda x.M, N)) = N$. Please note that this is just a sketch, not a well-thought out proposal about timed λ-calculi, because we have not addressed crucial questions like the effect of ϕ on $(\lambda x.M)$, reduction strategies or the closure of $\rightarrow$ under λ-abstraction or timer formation. One would also have to address idling. Do we want $M \rightarrow \phi(M)$ or progress assumptions? How do reduction strategies live with progress assumptions? Etc ...

The point of this excursion was to illustrate just how easy it is to add (discrete) timing. All one needs is a (small-step) reduction relation (which immediately raises the question of discrete timing vs. large step operational semantics).

Semantics?

The semantics of computational formalisms can be specified in many ways, but with respect to timing, much existing work seems to have been done in an operational setting, where the bifurcation between probabilistic and non-probabilistic semantics is pertinent.

One way of thinking about the transformation from the asynchronous π-calculus to π_t is that it involves two things.

- The addition of a timer.
- Assignment of durations to the computational process, by adjoining a duration map from $\rightarrow$ or $\twoheadrightarrow$ into the time domain.

In this chapter, we have used what must be the simplest duration map by making every reduction step consume one unit of time. Alternatively, one could use a less trivial map. The time domain need not be $\mathbb{N}$ and durations could vary. For example, remote communications could take longer than local ones. A more radical change would be to have a duration relation rather than a function. The resulting non-determinism can be helpful for obtaining upper or lower bounds for durations of computations. Assigning probabilities to the durations proscribed by the duration

relation would be a further generalisation. This makes more sense if the rest of the semantics is also probabilistic. Our reason for not using probabilistic semantics is simplicity. An overview of probabilistic timed process calculi can be found in [18].

Time Flow?

Another crucial issue is how to model the flow of time, or more precisely, how to distribute information about the flow of time throughout the process. Our approach is to use *implicit broadcasting*, implemented with timesteppers. There are three main reasons for our choice. Firstly, it is different from existing approaches: the more, the merrier. Secondly, it seems closest to "reality": existing parallel computers just don't need communication or synchronisation to pass time (although this conviction may just be the result of the present author's misunderstanding of physical time). Finally, implicit broadcasting works just as well with reduction semantics as it does with labelled transitions. This is an advantage because it is often harder to come up with the latter than the former. A case in point is the Ambient Calculus [25] which, in 2002, is still lacking a comprehensive labelled semantics. The reasons why other models of timed computation use labelled semantics are probably largely historic: reduction semantics for concurrency emerged much later than labelled transitions, and when they did, a veritable amount of technical development of labelled models of timed computation had already been accumulated and exerted gravitational pull.

To understand the differences and similarities between the two approaches, let us consider an example of timed labelled transitions. Typically, such a model would feature *time passing transitions* in addition to the usual communications. The former would be $\mathsf{P} \xrightarrow{r} \mathsf{Q}$ where r is an element of the time domain, for example a real number above zero. The transition should be read as saying that while r units of time pass, the process evolves from P to Q. An important special case is idling of processes without active timers: $\mathsf{P} \xrightarrow{r} \mathsf{P}$. The flow of time is synchronised by broadcasting.

$$\frac{\mathsf{P} \xrightarrow{r} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{r} \mathsf{Q}'}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{r} \mathsf{P}' \mid \mathsf{Q}'}$$

A source of variation is the integration of time passing with interaction. If $\mathsf{P} \xrightarrow{\overline{x}\langle y\rangle} \mathsf{P}'$ and $\mathsf{Q} \xrightarrow{3.21388642} \mathsf{Q}'$, for what l can do we have $\mathsf{P} \mid \mathsf{Q} \xrightarrow{l} \mathsf{P}' \mid \mathsf{Q}'$? A common approach for answering such questions is to assign durations (possibly being 0) to all actions and display them in transitions: $\xrightarrow{\overline{x}\langle y\rangle,2.7}$ instead of $\xrightarrow{\overline{x}\langle y\rangle}$. The (Par) then become something like

$$\frac{\mathsf{P} \xrightarrow{l,t} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{l',t} \mathsf{Q}' \quad \ldots}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{l'',t} \mathsf{P}' \mid \mathsf{Q}',}$$

where where we have omitted the details of when and how to get l'' from l and l' (for simplicity, we identify the idling transition $\xrightarrow{r}$ with $\xrightarrow{r,r}$).

Integration of Time and Computation?

It is possible to add durations (or durations plus probabilities) to a model of computation but nothing else. Degano, Lobbo and Priami have done this for the π-calculus [32]. The resulting formalisms are useful for reasoning about execution times of the algorithms in the underlying untimed calculi. This can be very fruitful and all of complexity theory [89], at least where it studies time-complexity, is a special case where the underlying untimed formalism is Turing Machines and every step in the computation takes one unit of time.

Just adding durations lacks the ability to make time a first class citizen of the computational process. The limits of what that may mean have probably not been explored so far, but we surely would like to be able to make computation contingent upon the flow of time. This can be achieved in various ways and timers appear to be the most versatile tool to do so, but there are alternatives. One is *delay operators* which wait for a predetermined period of time before they launch a chosen process. *Watchdogs* are also popular: they launch a process every n units of time. In §4.3.2 we have shown how our timer can implement delay operators and watchdogs.

But what about our timer? Is there much scope for variations in its definition? Two things come to mind. Firstly, one could stop requiring that the time-in continuation was input guarded and allow $\mathsf{timer}^t(\mathsf{P},\mathsf{Q})$ instead of just $\mathsf{timer}^t(x(\vec{v}).\mathsf{P},\mathsf{Q})$. The (TimeIn) rule would have to be modified accordingly (here in its labelled variant).

$$\frac{\mathsf{P} \xrightarrow{l} \mathsf{P}'}{\mathsf{timer}^t(\mathsf{P},\mathsf{Q}) \xrightarrow{l} \mathsf{P}'.}$$

We advise against this change, despite its syntactic simplicity. Just like unrestricted sums, having unguarded timers would be hard to implement on existing computers and would lead to counterintuitive behaviour without contributing anything useful. Secondly, one feature of our timer is that time-out times are determined statically, where dynamic generation would be desirable. It may sometimes be useful to have processes like $x(t).\mathsf{timer}^t(y(\vec{v}).\mathsf{P},\mathsf{Q})$. In a pure π-calculus, this can be simulated by branching: $x[\&_{t>0}\ ().\mathsf{timer}^t(y(\vec{v}).\mathsf{P},\mathsf{Q})]$. Alternatively, we would have to use what may be called an "applied π-calculus" such as Pict [91] that has integers as a basic data structure. Neither approach is problematic.

Progress Assumptions?

As already discussed in §4.3.1 and §4.7.3, progress assumptions constrain idling. The two limit cases are the popular options: not restrict idling at all, or requiring maximal progress, where idling is only allowed when no other type of transition can be inferred. We do not wish to adjudicate further on progress assumptions, they all have their traditions of justification.

4.8.1 Three Examples

We conclude this – so far – somewhat theoretical account of some of the main design choices for timed calculi by presenting three exemplary timed calculi, that have been discussed in the literature. The first two, Real-Time CCS and RtCCS, are rather similar to π_t, in particular, they are based on CCS, π's predecessor. Both use dedicated time passing action, but RtCCS has a discrete time domain without progress assumptions, while Real-Time CCS is parameterised by a time domain. We finish this overview with explaining the main ideas behind Timed Automata, a very general and popular formalism build on top of finite state automata.

RtCCS

RtCCS [105] was proposed by Satoh and Tokoro, with the aim of working towards better models for DS. The presentation of RtCCS is essentially the same as that of CCS, except for the *tick action* $\surd$ and the *timer* $\mathsf{timer}^t(.,.)$. Processes are then given by the following grammar.

$$\mathsf{P} ::= 0 \mid X \mid \alpha.\mathsf{P} \mid \mathsf{P}+\mathsf{Q} \mid \mathsf{P}\,|\,\mathsf{Q} \mid \mathsf{P}[f] \mid \mathsf{P}\setminus L \mid \mathsf{rec}\,X:\mathsf{P} \mid \mathsf{timer}^t(\mathsf{P},\mathsf{Q})$$

This definition uses the usual notions of names, co-names, τ-action etc. Actions are names, co-names or τ. In $\mathsf{rec}\,X:\mathsf{P}$, X is guarded. You'll find the operational semantics in Figure 4.5. We can see that RtCCS sets up its discrete time domain in the context of a conventional, not probabilistic labelled semantics, much like π_t. However, and here the differences start, time flow is propagated with an explicit time passing action $\surd$ that requires not only labelled transitions but also a large number of additional transition rules. RtCCS also enforces maximal progress because it permits to infer $\mathsf{P} \xrightarrow{\surd} \mathsf{Q}$ only if P cannot engage in τ-transitions. Of course it would be easy to remove this progress assumption. The most interesting feature of Satoh's and Tokoro's calculus is its timer. On the surface is seems quite like π_t's, but the details differ subtly, with significant semantic effects. Firstly, input-guards are not required. This permits all manner of pathologies such as the process $\mathsf{timer}^t(\tau.\mathsf{P},\mathsf{Q})$ which can stop itself: $\mathsf{timer}^t(\tau.\mathsf{P},\mathsf{Q}) \xrightarrow{\tau} \mathsf{P}$. It is also possible to simultaneously stop several nested timers at once:

$$\mathsf{timer}^t(\mathsf{timer}^t(\mathsf{timer}^t(x.\mathsf{P},\mathsf{Q}),\mathsf{R}),\mathsf{S}) \xrightarrow{x} \mathsf{P}$$

Another interesting design choice is to allow time to pass in a timer's time-in continuation. This means timers in the time-in continuation are activated as soon as the enclosing timer is. This can block timers that have not timed-out yet: as $\tau.\mathsf{P}$ cannot perform a $\surd$ action, because that would violate maximal progress, neither

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

$$\frac{\mathsf{P} \xrightarrow{\alpha} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{\overline{\alpha}} \mathsf{Q}'}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{\tau} \mathsf{P}' \mid \mathsf{Q}'} \qquad \frac{\mathsf{P} \xrightarrow{\alpha} \mathsf{P}' \quad \alpha \notin L \cup \overline{L}}{\mathsf{P} \setminus L \xrightarrow{\alpha} \mathsf{Q} \setminus L} \qquad \frac{\mathsf{P} \xrightarrow{\alpha} \mathsf{Q}}{\mathsf{P}[f] \xrightarrow{f(\alpha)} \mathsf{Q}[f]}$$

$$\frac{\mathsf{P}\{\mathsf{rec}\, X : \mathsf{P}/X\} \xrightarrow{\alpha} \mathsf{Q}}{\mathsf{rec}\, X : \mathsf{P} \xrightarrow{\alpha} \mathsf{Q}} \qquad \frac{\mathsf{P} \xrightarrow{\alpha} \mathsf{P}'}{\mathsf{timer}^{t+1}(\mathsf{P}, \mathsf{Q}) \xrightarrow{\alpha} \mathsf{P}'} \qquad \frac{\alpha \neq \tau}{\alpha.\mathsf{P} \xrightarrow{\surd} \alpha.\mathsf{P}}$$

$$\frac{\mathsf{P} \xrightarrow{\surd} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{\surd} \mathsf{Q}'}{\mathsf{P} + \mathsf{Q} \xrightarrow{\surd} \mathsf{P}' + \mathsf{Q}'} \qquad \frac{\mathsf{P} \xrightarrow{\surd} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{\surd} \mathsf{Q}' \quad \mathsf{P} \mid \mathsf{Q} \xrightarrow{\alpha} \mathsf{R} \Rightarrow \alpha \neq \tau}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{\surd} \mathsf{P}' \mid \mathsf{Q}'}$$

$$\frac{\mathsf{P} \xrightarrow{\surd} \mathsf{P}'}{\mathsf{P} \setminus L \xrightarrow{\surd} \mathsf{Q} \setminus L} \qquad \frac{\mathsf{P} \xrightarrow{\surd} \mathsf{P}'}{\mathsf{P}[f] \xrightarrow{\surd} \mathsf{Q}[f]} \qquad \frac{}{0 \xrightarrow{\surd} 0}$$

$$\frac{\mathsf{P}\{\mathsf{rec}\, X : \mathsf{P}/X\} \xrightarrow{\surd} \mathsf{Q}}{\mathsf{rec}\, X : \mathsf{P} \xrightarrow{\surd} \mathsf{Q}} \qquad \frac{\mathsf{P} \xrightarrow{\surd} \mathsf{P}'}{\mathsf{timer}^{t+1}(\mathsf{P}, \mathsf{Q}) \xrightarrow{\surd} \mathsf{timer}^{t}(\mathsf{P}', \mathsf{Q})}$$

$$\frac{\mathsf{Q} \xrightarrow{\alpha} \mathsf{Q}'}{\mathsf{timer}^{1}(\mathsf{P}, \mathsf{Q}) \xrightarrow{\alpha} \mathsf{Q}'} \qquad \frac{\mathsf{Q} \xrightarrow{\surd} \mathsf{Q}'}{\mathsf{timer}^{1}(\mathsf{P}, \mathsf{Q}) \xrightarrow{\surd} \mathsf{Q}'}$$

Figure 4.5: Transitions for RtCCS. Here α ranges over all actions apart from $\surd$ and $t \in \mathbb{N}$ is greater than 0.

can $\mathsf{timer}^1(..., \tau.\mathsf{P})$. Hence for no R

$$\mathsf{timer}^2(\mathsf{timer}^1(..., \tau.\mathsf{P}), \mathsf{Q}) \xrightarrow{\surd} \mathsf{R}.$$

This is odd because the outermost timer has clearly not timed-out yet. Even worse, the time-out continuation of the inner timer can spontaneously stop the outer timer:

$$\mathsf{timer}^2(\mathsf{timer}^1(..., \tau.\mathsf{P}), \mathsf{Q}) \xrightarrow{\tau} \mathsf{P}$$

The benefits of such behaviour, other than syntactic simplicity, are not entirely obvious.

One crucial difference between RtCCS and π_t is that computations in the former have no duration: the execution of a non $\surd$-action does not affect timers in any way. This allows to derive transitions like

$$\mathsf{rec}\, X : \overline{x}.X \mid \mathsf{timer}^7(\mathsf{P}, \mathsf{Q}) \xrightarrow{\overline{x}} \mathsf{rec}\, X : \overline{x}.X \mid \mathsf{timer}^7(\mathsf{P}, \mathsf{Q}).$$

Hence, we can output an arbitrary number of $\overline{x}$'s without $\mathsf{timer}^7(\mathsf{P}, \mathsf{Q})$ timing out. Moreover, if $\mathsf{rec}\, X : \overline{x}.X \mid \mathsf{timer}^7(\mathsf{P}, \mathsf{Q})$ runs in parallel with some consumer of all the outputs on x, the timer cannot time-out at all because maximal progress prevents idling. This is problematic. This combination of maximal progress and timers being unaffected by computation (as opposed to idling) makes it hard to see how RtCCS could be used to model error recovery in distributed systems where timers get started to allow a process to wait for a while for some remote message to come in, but if it doesn't, the timer times out and triggers error recovery. But if, while waiting for the remote message to come in, the process can do other computations, as it would in many DS, time-out could not happen, subverting the purpose of timer-based error detection and recovery.

Despite these differences, the similarities between RtCCS and π_t indicate that it would be easy to use dedicated time-passing actions like $\surd$ to give labelled semantics to π_t without using timesteppers. It would be just as straightforward to give timestepper based semantics, labelled or otherwise, to RtCCS. It appears not to matter much whether one chooses to implement the flow of time by dedicated time-passing actions or with timesteppers. These are just two ways of doing the same thing. In the absence of a convincing axiomatisation of discretely timed π-calculi, we cannot establish this formally, but it seems that there should be a straightforward translation between these two ways of implementing time flow. Whether this 'presentation independence' also holds for non-discrete time domains is less clear.

Real-Time CCS

Real-Time CCS, introduced by Yi in [122], also takes CCS as its launch pad, but, unlike the previous calculus, is parameterised by a time domain and hence not

$$\frac{}{0 \xrightarrow{\epsilon_t} 0} \qquad \frac{}{\alpha.\mathsf{P} \xrightarrow{\alpha} \mathsf{P}} \qquad \frac{\alpha \neq \tau}{\alpha.\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{P}} \qquad \frac{}{\tau.\mathsf{P} \xrightarrow{\tau} \mathsf{P}}$$

$$\frac{}{\epsilon_{s+t}.\mathsf{P} \xrightarrow{\epsilon_s} \epsilon_t.\mathsf{P}} \qquad \frac{}{\epsilon_t.\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{P}} \qquad \frac{\mathsf{P} \xrightarrow{\epsilon_s} \mathsf{Q}}{\epsilon_t.\mathsf{P} \xrightarrow{\epsilon_{s+t}} \mathsf{Q}}$$

$$\frac{\mathsf{P} \xrightarrow{\alpha} \mathsf{P}' \quad \alpha \neq \epsilon_t}{\mathsf{P}+\mathsf{Q} \xrightarrow{\alpha} \mathsf{P}'} \qquad \frac{\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{\epsilon_t} \mathsf{Q}'}{\mathsf{P}+\mathsf{Q} \xrightarrow{\epsilon_t} \mathsf{P}'+\mathsf{Q}'}$$

Figure 4.6: Transitions for regular agents in Timed CCS

restricted to discrete time steps. Its semantics, too, is given in form of a non probabilistic, maximally progressing extension with time-passing actions of the original CCS labelled transitions. Unlike in RtCCS, delay operators, not timers are used for making computation contingent on time-flow, but the combination of timers and unrestricted sums allows to implement a form of timers, as we describe below. But first a presentation of the relevant syntactic and semantic basics.

By a *time domain* $\mathcal{T}$, we mean an arbitrary well-ordered set. Its least element is denoted 0. The set $\{\epsilon_t \,|\, t \in \mathcal{T}\}$ contains all *empty actions*, where t is the *duration* of ϵ_t. Again, we have the usual notions of names, action etc, but since we have introduced many new actions, rather than just one, we shall be a less brief with the details than we were for RtCCS. Let $\mathcal{N}$ be a countably infinite set of names with $\tau \notin \mathcal{N}$. Then $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$ is the set of *labels* where $\overline{\mathcal{N}}$ is the set of *co-names*, i.e. if $x \in \mathcal{N}$ then $\overline{x} \in \overline{\mathcal{N}}$. τ and empty actions are their own complements: $\overline{\tau} = \tau$ and $\overline{\epsilon_t} = \epsilon_t$. An *action* is any member of $\mathcal{L}$, any empty action and τ. The *regular agents* are given by the grammar

$$\mathsf{P} ::= 0 \;\mid\; \alpha.\mathsf{P} \;\mid\; \mathsf{P}+\mathsf{Q} \;\mid\; X$$

where X is an agent variable and α ranges over actions. *Regular agents* are defined by equations $(\mathsf{P}_i = X_i)_{i \in I}$. We omit further details. The labelled operational semantics of regular agents are given in Figure 4.6.

Regular agents have the following two properties.

THEOREM 26
- *(temporal determinacy) If* $\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{Q}$ *and* $\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{R}$*, then* $\mathsf{Q} \equiv \mathsf{R}$ *(we omit to define the structural congruence* $\equiv$*, as it is standard).*
- *(maximal progress) If* $\mathsf{P} \xrightarrow{\tau} \mathsf{Q}$ *then no* R, t *exist such that* $\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{R}$*.*
- *(temporal continuity)* $\mathsf{P} \xrightarrow{\epsilon_{s+t}} \mathsf{Q}$ *if and only if for some* R*:* $\mathsf{P} \xrightarrow{\epsilon_s} \mathsf{R} \xrightarrow{\epsilon_t} \mathsf{Q}$*.*
- *(temporal persistence) Whenever* $\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{Q}$ *and* $\mathsf{P} \xrightarrow{\alpha} \mathsf{R}$*, then* $\mathsf{Q} \xrightarrow{\alpha} \mathsf{S}$ *for some regular agent* S*.*

But what about parallel composition? One of the principles guiding the design was to have maximal progress. Unfortunately, the obvious rule

$$\frac{\mathsf{P} \xrightarrow{\epsilon_s} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{\epsilon_s} \mathsf{Q}'}{\mathsf{P} \,|\, \mathsf{Q} \xrightarrow{\epsilon_s} \mathsf{P}' \,|\, \mathsf{Q}'}$$

would destroy maximal progress as one can easily verify. But how to prevent agents running in parallel from idling when together, they can perform τ actions? In §4.7.3, we have discussed how to do that in π_t, but [122] goes a rather different path and uses *timed sorts*. Their definition proceeds in several steps. Let $\alpha \neq \epsilon_s$. Then

- $\mathsf{P} \xrightarrow{\alpha}_0 \mathsf{Q}$ iff $\mathsf{P} \xrightarrow{\alpha} \mathsf{Q}$.
- $\mathsf{P} \xrightarrow{\alpha}_t \mathsf{Q}$ iff $\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{R} \xrightarrow{\alpha} \mathsf{Q}$.

Let $L \subseteq \mathcal{L}$. A regular agent P has *sort* L *within* t if for all $t' < t$ and all $\alpha \in \mathcal{L}$: whenever there is Q with $\mathsf{P} \xrightarrow{\alpha}_{t'} \mathsf{Q}$, then $\alpha \in L$.

Let $s, t > 0$ and P be a regular agent. Then $\mathsf{sort}_t(\mathsf{P})$ is inductively given by the following clauses.

- $\mathsf{sort}_t(\mathsf{0}) = \emptyset$.
- $\mathsf{sort}_t(\alpha.\mathsf{P}) = \emptyset$, where $\alpha \neq \epsilon_s$.
- $\mathsf{sort}_t(\epsilon_t.\mathsf{P}) = \emptyset$.
- $\mathsf{sort}_t(\epsilon_{s+t}.\mathsf{P}) = \emptyset$.
- $\mathsf{sort}_{s+t}(\epsilon_t.\mathsf{P}) = \mathsf{sort}_s(\mathsf{P})$.
- $\mathsf{sort}_t(\mathsf{P} + \mathsf{Q}) = \mathsf{sort}_t(\mathsf{P}) \cup \mathsf{sort}_t(\mathsf{Q})$.
- $\mathsf{sort}_t(X) = \mathsf{sort}_t(P)$, assuming $X = \mathsf{P}$ is the defining equation for X.

It is easy to show that for each regular agent P and $t > 0$, $\mathsf{sort}_t(\mathsf{P})$ is a timed sort for P within t. If we abbreviate $\{\overline{\alpha} \,|\, \alpha \in L\}$ to $\overline{L}$ then we can show that for all regular agents P and Q: if $\mathsf{sort}_t(\mathsf{P}) \cap \overline{\mathsf{sort}_t(\mathsf{Q})} = \emptyset$, then for all $t' < t$: $\neg\exists \mathsf{P}', \mathsf{Q}'.\mathsf{P} \xrightarrow{\alpha}_{t'} \mathsf{P}'$, $\mathsf{Q} \xrightarrow{\alpha}_{t'} \mathsf{Q}'$. This fact justifies the rule for parallel composition in Real-Time CCS.

$$\frac{\mathsf{P} \rightarrow \mathsf{P}' \quad \alpha \in \mathcal{L} \cup \{\tau\}}{\mathsf{P} \,|\, \mathsf{Q} \rightarrow \mathsf{P}' \,|\, \mathsf{Q}} \qquad \frac{\mathsf{P} \xrightarrow{\alpha} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{\overline{\alpha}} \mathsf{Q}' \quad \alpha \in \mathcal{L}}{\mathsf{P} \,|\, \mathsf{Q} \xrightarrow{\tau} \mathsf{P}' \,|\, \mathsf{Q}'}$$

$$\frac{\mathsf{P} \xrightarrow{\epsilon_t} \mathsf{P}' \quad \mathsf{Q} \xrightarrow{\epsilon_t} \mathsf{Q}' \quad \mathsf{sort}_t(\mathsf{P}) \cap \overline{\mathsf{sort}_t(\mathsf{Q})} = \emptyset}{\mathsf{P} \,|\, \mathsf{Q} \xrightarrow{\epsilon_t} \mathsf{P}' \,|\, \mathsf{Q}'}$$

It can be shown that $\mathsf{P} \mid \mathsf{Q}$ satisfies all four properties in Theorem 26 for all regular agents P and Q. (Real-Time CCS has two more constructors familiar from CCS: renaming and restriction. We omit their details.) Next we show how [122] implement proposes to implement timers in Real-Tim CCS. Here is the definition.

$$\mathsf{timer}^t(x.\mathsf{P}, \mathsf{Q}) = x.\mathsf{P} + \epsilon_t.\mathsf{Q}$$

Now clearly

$$\mathsf{timer}^t(x.\mathsf{P}, \mathsf{Q}) \xrightarrow{\epsilon_t} \mathsf{Q} \qquad \mathsf{timer}^t(x.\mathsf{P}, \mathsf{Q}) \xrightarrow{x} \mathsf{P},$$

so timer-like behaviour is definitely possible. Unfortunately this timer may also not time-out in time. Here's how. Define $\mathsf{Q}_t = \epsilon_t.\mathsf{R}$. Then

$$\dfrac{\dfrac{\dfrac{}{\mathsf{Q}_3 \xrightarrow{\epsilon_2} \mathsf{Q}_1}}{\epsilon_4.\mathsf{Q}_3 \xrightarrow{\epsilon_6} \mathsf{Q}_1} \qquad \dfrac{}{x.\mathsf{P} \xrightarrow{\epsilon_6} x.\mathsf{P}}}{\mathsf{timer}^4(x.\mathsf{P}, \mathsf{Q}_3) \xrightarrow{\epsilon_6} x.\mathsf{P} + \mathsf{Q}_1 = \mathsf{timer}^1(x.\mathsf{P}, R)}$$

While it is probably possible to concoct a justification for such behaviour, we prefer to consider it an anomaly. The problem is caused by the temporal semantics of the unrestricted sum, or, more specifically, by allowing time to pass in the summands. Whether it is possible to encode a timer exactly like π_t's is unclear.

4.8.2 Timed Automata

Timed Automata [6] are currently the most widely used formalism for modelling timed computation. They take conventional deterministic or non-deterministic finite automata as underlying computational mechanism and extend them with clocks and clock-constraints that range over $\mathbb{R}^+$, but that does not mean they have a continuous time domain. As they are quite different from process theoretic models, we state their definition first.

Definition 53 [27] Let C be a set of *clock variables*, each ranging over $\mathbb{R}^+$. The induced *clock constraints over* C, denoted $\mathsf{CC}(C)$, are given inductively by the following rules.

- If $c \in \mathbb{Q}^+$ and $cv \in C$, then $cv < c$, $cv \leq c$, $cv > c$, $cv \geq c$ are all clock constraints.
- If cc_1, cc_2 are clock constraints, then $cc_1 \wedge cc_2$ is a clock constraint.

A *timed automaton* consists of

- a finite *alphabet* Σ.
- A finite set S of *states*.

- A set $S_0 \subseteq S$ of *start states*.
- A set C of clock variables.
- A map i from S to $\mathsf{CC}(C)$. We call i the *invariants*.
- The transition relation $\rightarrow$ which is a subset of $S \times \Sigma \times \mathsf{CC}(C) \times \mathcal{P}(C) \times S$.

A transition $s \xrightarrow{l,c,R} t$ means that in state s, the automaton can accept the label l to evolve to state t, provided that the constraint c is met. When the transition is executed, after it has left s, but before it reaches t, all clocks in R are reset to 0.

Interestingly, bare timed automata do not on their own have a notion of time. That is separated from the model and hidden in the structures they accept. But what do they accept? Conventional automata accept words, but a timed automaton $(\Sigma, S, S_0, C, \rightarrow)$? They accept certain infinite transition systems, $(\Sigma, S', S'_0, \rightarrow')$ that can be seen as generalisations of words to *timed words.* Here S' is a set of states, each of which is a pair (s, ca), where $s \in S$ and $ca : C \rightarrow \mathbb{R}^+$ is a *clock assignment* which assigns to each clock its *current time.* $S'_0 = \{(s, ca) \mid s \in S_0, \forall c \in C.ca(c) = 0\}$ are the start states of the transition system, where all clocks read 0. Transitions $\rightarrow'$ are either

$$(s, ca) \xrightarrow{d}{}' (s, \lambda c.ca(c) + d) \qquad \text{or} \qquad (s, ca) \xrightarrow{l}{}_a (s', ca').$$

We call $(s, ca) \xrightarrow{d}{}_d (s, \lambda c.ca(c) + d)$ a *delay transitions* and require that for all $0 \leq d' \leq d$ the invariant i holds for $\lambda c.ca(c) + d'\ d \in \mathbb{R}^+$. For *action transitions* $(s, ca) \xrightarrow{l}{}_a (s', ca')$, we it must be the case that $l \in \Sigma$ and the timed automaton has a transition $s \xrightarrow{l,c,R} t$ such that ca satisfies c and $ca' = \lambda c'.\texttt{if}\ c' \in R\ \texttt{then}\ 0\ \texttt{else}\ ca(c')$

Usually one imposes additional conditions, for example that the automaton must be *non-xeno*. That means, it is not permitted to have an infinite number of transitions in finite time. In addition, we require that it must be possible for automata to *progress to infinity*. That means, for each state the constraints must either allow the computation to stay at the state forever or the constraints on transitions from that state must permit the computation to execute at least one of them.

It is clear from this definition that timed automata do not enforce progress assumptions and don't feature probabilistic semantics. Since timed automata do not have structured states, unlike process calculi, the question of time flow from one part of the computation to another does not really make sense. In summary, timed automata are very different kinds of models compared to process calculi, because the latter contain an algebra of programs which is simply abstracted away in timed automata. The well-known limited expressivity of the underlying untimed formalism prevents Timed Automata from being a serious contender for DS. It would nevertheless be possible to implant their approach to timing (clock constraints, timed words, etc.) in other models, in particular the asynchronous π-calculus.

4.9 Concluding Remarks

This chapter has introduced π_t, the asynchronous π-calculus with timers and explored some of its basic properties. Extensions are possible in many directions. We have already devoted §4.7 to what may be the most important next step, finding coarser equivalences. Another area deserving further investigations is expressivity. There is good reason to believe that π_t cannot have a good encoding into untimed calculi, although a definite result has proven elusive. What about the other way round? Can the asynchronous π-calculus be coded up convincingly in π_t? Chapter 2 spoke about the many existing encoding into π-calculi. Do they also hold in π_t? It would also be nice to see how π_t could deal with dense or continuous time domains, or assignments of probabilities to durations. This could lead to a connection between π-calculi and hybrid systems. And what about separation results, such as Palamidessi's about the non-encodability of mixed choice into the asynchronous π-calculus [88]? Clearly, the timer is a form of mixed sum, but is it expressive enough to code up the unrestricted choice? The encodability of sums by timers, but also the very possibility of λ_t touches on an interesting issue: does the existence of a timer construct alone make a calculus concurrent? If timers can encode non-determinism without reservation, then the answer must be to the affirmative because parallel composition can be expanded into summation [74]. But it seems that even in the more likely case where sums cannot be encoded, some concurrency-like phenomena would be present. We feel that there is a deep connection between time-flow, the spatial extension of the computational process and concurrency, that may not have been fully understood as yet.

We conclude this chapter with some philosophical speculations about the Church-Turing Thesis and its relation to timed computation. Is is independent from – although deeply influenced by and influencing of – the rest of this text and either can be read without the other.

4.10 The Church-Turing Thesis and Timed Computations

Consider the following program.

```
every 31 seconds
{
        print "looking at my Gucci, it's about that time!"
}
```

Our contentions are simple.

1. This program is mechanically computable, hence it must be expressible in every model of mechanical computability.

2. One cannot verify that a given model of computation allows to express programs such as the above unless elementary steps of computations are assigned durations in the model. In particular, formalisms such as π-calculi, Turing Machines or μ-recursive functions do not allow this verification and can thus not express the program above.

3. Assigning durations to computations strictly extends conventional models of computation.

But let's begin at the beginning ...

The Church-Turing Thesis [28, 51] is one of the most famous and fundamental conjectures of the whole of computing theory. Its precise meaning and epistemological status are controversial but its validity is rarely doubted. To simplify matters, we shall avoid the problem of Church's and Turing's original intentions. We are interested in computation, or rather the limits of computation, not history. In our reading, the Church-Turing Thesis asserts that computation is a phenomenon with sharp conceptual boundaries, sharp enough in fact, to allow convincing mathematisation, for example by way of Turing Machines or π-calculi. Our contention above, then, may be summarised as claiming that the conventional demarcation of computation given by the Church-Turing Thesis, is too restrictive and ought to be liberalised.

When we speak of the Church-Turing Thesis, we have statements like the following in mind.

> *The process of interaction with an environment, exchanging finite data in each interaction, by a mechanical process with a finite program of instructions, in accord with the laws of physics, apart from resource constraints, can be precisely simulated by a π-calculus process.*

Several issues are worth noting. Firstly, the relation "simulating computation" is one between a mathematical formalism and something informal, intangible, such as physical processes. Hence it seems in principle impossible to mathematically decide if a given formalism models computation. Secondly, although there are good reasons to prefer interaction-based formalisms, the reference to the π-calculus could be replaced by other models such as Turing Machines or μ-recursive functions. Finally, it is left open what it means for a formalism to be "precisely simulated".

The first two points, while raising interesting issues, do not concern us here. It is the last point that is found wanting once one thinks seriously about *timed* computation. The problem is not that incorporating some notion of time into a model of computation would allow to solve the halting problem, although we have

not seen a proof that such a drastic extension of computational power is impossible. The problem is that the received form of the Church-Turing Thesis excludes timed algorithms, such as our example above, tout-court from the realm of mechanical computability. Conventional models of computation simply do not have temporal properties. We propose to ameliorate this shortcoming by integrating the passing of physical time into models of computation. Extending the work of this thesis, a *real-timed π-calculus* would be the result, where each reduction step is assigned its duration, a real number greater than 0. The Church-Turing Thesis could then be rephrased.

> *The process of interaction with an environment, exchanging finite data in each interaction, by a mechanical process with a finite program of instructions, in accord with the laws of physics, apart from resource constraints, can be precisely simulated by a real-timed π-calculus process.*

Before delving into the details of our criticism, we would like to emphasise the speculative nature of this undertaking: we are neither fully convinced of its correctness nor can we provide the details of how an appropriate real-timed π-calculus would look like. In addition, one must distinguish two things:

- The problem under discussion here, whether models of computation ought to be augmented so as to enable them to deal with timed computation.

- The question of the truth or falsehood of the temporally extended Church-Turing Thesis. After all, the existence of universal models of computation may be peculiar to the world of untimed computation. Maybe timed computation is a fundamentally more vague concept.

The following points are also worthwhile to be borne in mind.

- We do not address the issue of just what kinds of duration assignments are legitimate. Would it be a good idea to allow the assignment of 7 seconds to every computational step in a terminating computation and 10 for all others? Wouldn't that allow a timed observer to solve the halting problem? It may be necessary to permit only computable (in the conventional sense) assignments. Another problematic duration assignment was proposed in the context of *Accelerating Turing Machines* [29]: the n^{th} step in the computation takes $\frac{1}{2^n}$ seconds. Accelerating Turing Machines can decide the halting problem in finite time, although taking an infinite number of steps in the process. Since all the available evidence suggests that Accelerating Turing Machines are not in accordance with the laws of physics, such *Xeno*-assignments should probably also not be admissible.

- Is it sufficient to extend conventional models of computation with a notion of duration to capture all of timed computation? This might not be the case. It could be necessary to integrate the passing of time more tightly with the computational process, for example by adding timers. We will discuss this problem only very briefly.

After this little excursion we will now defend and discuss the three claims above. Regarding (1), the strongest argument in favour of our little program's being mechanically computable is that it can easily be written with any modern programming environment on run-of-the-mill hardware. All that's required is a conventional computer and (maybe) a clock. Both are prototypical mechanical devices [68] and there does not seem to be a reason to believe that this is no longer true for their combination. If the Church-Turing Thesis wants to be taken seriously, it better allowed to express trivial and ubiquitous programs such as the initial example.

Our argument for (2) has two parts. First, if we assign durations to all computational steps, we can clearly decide (ignoring for simplicity the problem of the decidability of the assignment) if the model generates what we interpret to be appropriate temporal behaviour. Secondly, we could not think of a way of doing this verification without an assignment of durations.

In defence of (3), it is clear that there is no such assignment in conventional models of computations. Hence assignments do extend these models, at least set-theoretically.

We expect the following types of (related) reactions from defenders of the Church-Turing Thesis.

The "Old-School Recursion Theorist's" Reply: *Of course Turing Machines can simulate this algorithm. Its timing is an inessential detail that can and should be ignored. All that matters is the function being computed, not the timing of the computational process. After all, even normal untimed computation proceeds in physical time and space but we don't care about this when pondering the essence of computation.*

This objection may be the easiest to deal with. Timed algorithms abound in the very fabric of computation: for example network flow-control algorithms such as TCP [111] or various operating systems' schedulers [10] are implemented using timers and their temporal properties are vital to these algorithms. Of course the Old-School Recursion Theorist could object that such claims misunderstand what it means for an algorithm to function: the point of models of computation is to abstract away from pesky little details like actual execution time. Well, maybe, and in case of the computation of, say, the factorial function it might be reasonable to ignore how long its computation takes. Why? Because we get something else: a number as a hard and fast result. We can "measure" the correctness of the algorithm by looking at this result. In the case of network flow-control algorithms

or OS schedulers, this is very different. The correctness of any implementation seems directly and indissolubly connected with temporal properties. Crudely put, a scheduler is something that guarantees my processes access to the CPU at least every 250 milliseconds. TCP ensures certain rather involved ratios between network capacity and transmission speed. It makes little sense to talk about the correctness of a scheduler or TCP without mentioning time. To appreciate the significance of the temporal aspects involved, consider the following transformation of sequential composition.

$$[\![\mathsf{P} \ ; \ \mathsf{Q}]\!] = [\![\mathsf{P}]\!] \ ; \ \texttt{sleep(}\ \texttt{1 year}\ \texttt{)} \ ; \ [\![\mathsf{Q}]\!]$$

If timing was an inessential detail to TCP or OS schedulers, we could apply this transformation without changing their semantic essence. Unfortunately, applying $[\![\cdot]\!]$ to all the world's C programs would render the entire planet's computational infrastructure humanly unusable. While that is an irrelevant detail sub speciae aeternis, it appears flippant to say that the intended semantics of such programs remains unaffected, at least for humans with an average lifetime of 54 years.

The Old-School Recursion Theorist might now reply that one can say the same thing about factorial functions: we could not humanly evaluate $!n$ for most n, if we'd apply the transform. But does that not suggest that timing is relevant even for factorials?

The "Separate Physics and Computation!" Reply: *It does not matter if Turing Machines can simulate timed computation or not because timed computation is not about pure algorithms. They are interactions of pure algorithms with physical devices, in this case clocks. The Church-Turing Thesis is only concerned with pure computation. The addition of clocks is no more relevant than the possibility of having computers extended with loudspeakers to produce sound or with wheels to allow physical movement.*

We could just say: "fair enough, but if timed computation is not computation, it is nevertheless something closely related and at least as interesting. It is also worth of mathematisation and hence of a timed equivalent of the Church-Turing Thesis."

But this is too conciliatory. Let's look at the defender's argument in more detail. It is a variant of the first objection and based on a distinction between (models) of pure algorithms and physical devices, the implication being that the latter have no role to play in the description of the former. This is problematic for two reasons. Firstly, ultimately every actual computation is a physical process and many models of computation idealise these physical processes and devices to some degree. An example would be the tape and head of a Turing Machine. So it cannot be the inclusion of models of physical entities in formalisms for computing that is deemed problematic, it is the inclusion of a specific kind of physical entity: the clock. But, and secondly, every model of computation (that we can think of) uses some form

of sequentialisation: "this must happen, then that", clearly a temporal property. Even domain-theoretic models use fixpoint iteration [48] that is usually imagined as a discrete, temporal process. It seems fair to say that time-stepping or discrete sequencing is intrinsic in all models of computation already. What is missing in most models is an explicit *duration* of the computational step. Once durations are specified, questions such as: "does this program implement that algorithm which has these temporal properties?" seem natural. If these temporal properties never mattered, we would be justified to just have a generic notion of discrete sequencing in the models, as, for example, given by the reduction steps in λ-calculi or Turing Machines. But, as we have argued above, the temporal properties do matter a lot in many situations, so it seems appropriate to model them explicitly because they are not induced by interaction with some arbitrary physical device, they come from the physical behaviour of something essential to computation itself: the flow of time.

Another problem with "Separate Physics from Computation!" is that our initial example does not actually require interaction with a clock, although in practise, implementations will. Conventional CPUs are constructed such that their computational steps are executed within tight time bounds. It may be possible to use these bounds to achieve the required temporal behaviour by judicious choice of translation into machine code alone. Of course these time bounds are almost always achieved with a clocking mechanism that "drives" the CPU. But this mechanism is not *explicit* in the code being executed. It is also possible, although currently unusual, for CPUs to lack an explicit clocking mechanism. That does not mean however, that computations executed on such devices have no temporal properties. It is just that these properties emerge from the physical properties of the executing hardware in a way that observers describe as clockless. Whether this is useful terminology touches on interesting problems (summarised by the question "what is a clock?"), that we do not wish to discuss further here.

The "That's what I'm Saying!" Reply: *Of course Turing Machines can simulate these algorithms. Just assume one step of a Turing Machines corresponds to one unit of time of the algorithm. In other words, conventional models of computation implicitly assign unit time, normalised to 1, to each computational step.*

Yes, but this assignment needs to be done. Why pretend it is not part of the computational model? This answer assumes an assignment, it is just not honest about it.

It also suffers from a more serious defect. It is not enough to simply fix an assignment once and for all, because every such assignment induces a minimal granularity of time and hence excludes certain more finely timed algorithms. If, for example, we'd assign 47 seconds as the duration of every computational step, our initial example would not be computable. Unless there is convincing empirical evidence of a physical limits to time-divisibility relevant to computation, none should be enforced

by our models of computation.

OK, Now We Have Durations, Are We Done Yet?

The arguments adumbrated above compellingly suggest to include a notion of duration into models of computation. But is that all we need to capture timed computation? Consider the following variant of our initial example.

```
on 14:Jan:1986 at 14:01:0221 GMT
{
        print "looking at my Gucci, it's about that time!"
}
```

Some, but not all of our arguments can be adapted to support the additional inclusion of an absolute notion of time into models of computation to accommodate this example.

Even ignoring the problem of absolute time and its relation to computation, is our augmented Church-Turing Thesis expressive enough? As already alluded to, many timed algorithms implement their temporal behaviour with the help of timers. Is it always possible, just armed with conventional models of computation augmented with durations, to simulate behaviour induced by the presence of timers? We are not sure. It might be possible to use busy-waiting to simulate timer driven interrupts, especially when there are no lower limits on the granularity of time steps and if the criteria that would warrant speaking of busy-waiting allowing to simulate timers, allow for some imprecision ...

This raises another issue: what does it mean to observe a timed computation? Should there be limits to observer's time-measuring abilities? Relatedly, is it appropriate to assign *exact* durations to computational steps? Wouldn't intervals or probabilities be better? Questions, questions ...

A Modest Proposal for a Research Programme

As pointed out, the Church-Turing Thesis is an empirico-philosophical assertion, not a mathematical one and cannot be verified or contradicted by purely formal means. Nevertheless, mathematical theorems can serve as evidence for or against its acceptance, the various mutual embeddability results of classical recursion theory being a prime example.

What kind of theorem could be significant evidence for or against the temporally extended Church-Turing Thesis? How about the following conjecture?

CONJECTURE 1 Let π_a be the asynchronous π-calculus and $\approx_a$ one of its reasonable equivalences, such as reduction-congruence or weak trace-equivalence. Let π_a^d be the asynchronous π-calculus extended with a reasonable notion of duration and $\approx_a^d$ one

of its reasonable equivalences. Then, in general, there is no mapping $[\![\cdot]\!]$ from π_a^d to π_a such that

$$[\![\mathsf{P}]\!] \approx_a^d [\![\mathsf{Q}]\!] \qquad \text{iff} \qquad \mathsf{P} \approx_a \mathsf{Q}.$$

This conjecture is attractive not only because despite its simplicity, establishing its truth value appears difficult, but also because it assumes little apart from duration assignment, essentially only the availability of appropriate notions of equivalence. This should not be a surprise given, on one hand, our preceding discussion of the Church-Turing Thesis with its subtext "what does it mean to observe a timed computation?", and the tight connection between notions of observation and equivalence on the other. Indeed, it does not appear to be an exaggeration to say that our a solution of our problem of timed computation would be but a special case towards answers for the following two questions. What does it mean to observe computation? When are two computations equal?

Chapter 5

Message Loss

This chapter adds a notion of location and two forms of simple message failure, message loss and message duplication, to the timed, asynchronous π-calculus and studies some of the basic equational properties of the resulting calculus.

5.1 Introduction

One of the main uses of timers is to unblock computations after they became stuck due to some fault such as a lost message or a deadlock. This is inconvenient to model in π_t because it lacks message failures and a proper notion of resource that could exhibit circular ownership patterns, a necessary precondition for deadlocks [30]. To explore timers in a more realistic setting, this section augments π_t with locations and non-byzantine message failure, obtaining π_{mlt}.

It is customary to distinguish four kinds of message failure.

- *Message loss*, which consists of a message emitted by some process to disappear without a trace. A naive incorporation of message loss could be achieved by adding the rule $\overline{x}\langle\vec{y}\rangle \rightarrow 0$. The chief cause for message loss is network saturation: a router has received more packets than it can store before passing them on and the surplus is simply discarded. In some flow control protocols, most prominently TCP, messages are explicitly discarded to communicate "please slow down!" to senders [111].

- *Message duplication* allows a message to be received more often then it was sent. A first attempt at modelling this phenomenon might be to have an additional rule $\overline{x}\langle\vec{y}\rangle \rightarrow \overline{x}\langle\vec{y}\rangle \mid \overline{x}\langle\vec{y}\rangle$. Message duplication is often a consequence of retransmissions after an acknowledgement message has been lost or arrived too late. Hence duplication and loss of messages are closely connected.

- If a message is received in a form that differs from how it was sent, we speak of *message corruption*. The rule $\overline{x}\langle\vec{y}\rangle \rightarrow \overline{x}\langle\vec{z}\rangle$, or even $\overline{x}\langle\vec{y}\rangle \rightarrow \overline{a}\langle\vec{z}\rangle$ expresses this

behaviour. Imperfections of the network infrastructure, but also malicious agents ("hackers") cause message corruption.

- We have an instance of *message forging* if a message was received that has never been sent *by a legitimate sender*: $\mathsf{0} \rightarrow \overline{x}\langle\vec{y}\rangle$. It is mostly caused by the aforementioned malicious agents.

Message forging and corruption are collectively referred to as *byzantine* failures. We have chosen to omit byzantine failures for three reasons. Firstly, they are much harder to account for. Secondly, non-byzantine faults are already fairly challenging, and thirdly, byzantine faults are less ubiquitous: any TCP based network must deal with message loss and duplication because they are part of that protocol's *normal* behaviour. Byzantine faults, on the other hand, can often be so infrequent as to be effectively absent (message corruption can easily be spotted through conventional redundancy and error-correction techniques [111], while separation from public access networks prevents message forging rather effectively). In the long run it will be unavoidable to extend our programme to include byzantine failures. It's just that we want to do the easy things first and leave the real work for later ...

Incorporation of non-byzantine message failure by just adding $\overline{x}\langle\vec{y}\rangle \rightarrow \mathsf{0}$ and $\overline{x}\langle\vec{y}\rangle \rightarrow \overline{x}\langle\vec{y}\rangle \mid \overline{x}\langle\vec{y}\rangle$ results in a calculus that is essentially unusable. Much of the communications between processes, indeed the majority, is usually considered reliable, in particular when it does not "go over then net". This suggests to distinguish reliable from unreliable communication in models. Abdulla and Jonsson propose to have two kinds of channels, non-dependable ones and dependable ones [4]. However, in distributed systems the same channel could be both reliable and unreliable depending on whether it is carrying a message sent locally or from a remote location. We take the remote/local binary as basic and augment the calculus with *sites* or *locations*, to allow to distinguish between local interactions that are not subject to message failures and remote communications that are. Sites have the form $[\mathsf{P}]_A$ where P is a π_t process and A is the associated set of *access points*, the names that P may receive data on. Sites can be composed in parallel and their names can be restricted. Messages travelling between locations that have left their originating site but have not yet arrived at their destination, that are "on the net", to use the vernacular once more, are called *in transit* or *in the ether*. Message failure can only occur while in transit. Of course similar concepts have been proposed before [8, 9, 14, 39, 99, 108, 121]. The main contribution of this chapter is to explore locations and timing together. Chapter 7 will show that location has other uses too.

Before going into details, we summarise our main design objective for this chapter.

- Addition of location and message failures should make minimal demands on

the calculus to be located. Distribution should accommodate as many non-distributed calculi as possible.

- The extension should be *incremental*. As much as possible of the equational theory and reasoning technology for the underlying non-distributed process calculus should be retained. Reasoning about π_{mlt} should be *modular*: we would like to separate reasoning about processes from reasoning about networks as much as possible. In particular, if two processes P and Q are equated by the underlying process calculus, then $[\mathsf{P}]_A$ and $[\mathsf{Q}]_A$ should also be equal in π_{mlt}.

- The extension should be as simple as possible while still allowing to exhibit the crucial phenomena that make distributed programming over the non-malicious networks difficult, even when byzantine failures are not an issue. In particular, we do not seek to model security protocols.

5.2 The Calculus

The calculus is a two sorted algebra, with one sort being *processes*, and the other *networks*, ranged over by $\mathsf{M}, \mathsf{N}, ...$ As processes, we take the timed asynchronous π-calculus introduced in the previous chapter. We repeat its syntax.

$$\mathsf{P} ::= x(\vec{y}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \mid \mathsf{P}|\mathsf{Q} \mid (\nu x)\mathsf{P} \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \mid !x(\vec{v}).\mathsf{P} \mid 0$$

Networks are given by the following grammar.

N	$::=$	$\overline{x}\langle\vec{y}\rangle$	message in transit
	$\mid$	$[\mathsf{P}]_A$	located process
	$\mid$	$\mathsf{M} \mid \mathsf{N}$	parallel composition of located processes
	$\mid$	$(\nu x)\mathsf{N}$	restriction
	$\mid$	0	inactive network

It will become obvious that π_t is not the only possible choice for processes to be located. Networks and their semantics require little from the underlying processes apart form having inputs and outputs. We do not explore alternatives to the choice of π_t for processes further here.

In the introduction to this chapter, we mentioned the access points A in $[\mathsf{P}]_A$ that contain the set of names that $[\mathsf{P}]_A$ might input on. But what does that mean? Consider the network $\overline{x}\langle a\rangle \mid [x(v).v(w).\mathsf{P}]_A$. Is a one of the names that $[x(v).v(w).\mathsf{P}]_A$ can receive messages on? One could find arguments either way, but we choose to circumvent the problems arising from both answers by simply prohibiting input-bound inputs. All located processes must be local (cf. §2.7.1). A formal definition of locality follows.

DEFINITION 54 The set of *local* π_t-processes is given by the following rules.

- 0 and $\overline{x}\langle\vec{y}\rangle$ are local.
- If $\{\vec{v}\} \cap \mathsf{fin}(\mathsf{P}) = \emptyset$, then $x(\vec{v}).\mathsf{P}$ and $!x(\vec{v}).\mathsf{P}$ are local.
- If $x(\vec{v}).\mathsf{P}$ and Q are local, then $\mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q})$ is local.
- If P and Q are local, then $\mathsf{P} \mid \mathsf{Q}$ is local.
- If P is local, then $(\nu x)\mathsf{P}$ is local.

As already alluded to in §2.7.1, locality does not significantly reduce the expressivity of π_t, but how realistic is the resulting calculus? In the Internet, hosts cannot dynamically receive *arbitrary* IP addresses and then use it to receive data, because messages would not be routed properly. This means in essence that programs utilising the Internet are local.

A related problem is whether to allow different sites to receive data on the same name, as in

$$[x(\vec{v}).\mathsf{P}]_{A\cup\{x\}} \mid [x(\vec{v}).\mathsf{Q}]_{B\cup\{x\}}.$$

We have decided against it, for simplicity. To have chosen the other alternative, would probably have made little substantial difference. Once routing concerns will also be important, the question must be reconsidered.

These well-formedness conditions on processes and networks are summarised by the next definition.

DEFINITION 55 We say N is *well-formed*, written $\vdash \mathsf{N}$, if $\vdash \mathsf{N}$ is derivable using the following rules.

- $\vdash \mathsf{0}$ is always derivable.
- $\vdash [\mathsf{P}]_A$ if P is local and each free input subject in P is in A.
- $\vdash \mathsf{N}_1|\mathsf{N}_2$ if $\vdash \mathsf{N}_1$ and $\vdash \mathsf{N}_2$ and, moreover, $\mathsf{ap}(\mathsf{N}_1) \cap \mathsf{ap}(\mathsf{N}_2) = \emptyset$.
- $\vdash (\nu x)\mathsf{N}$ if $\vdash \mathsf{N}$.

where the *access points* $\mathsf{ap}(\mathsf{N})$ of a network N are given by: $\mathsf{ap}([\mathsf{P}]_A) = A$, $\mathsf{ap}(\mathsf{N}_1|\mathsf{N}_2) = \mathsf{ap}(\mathsf{N}_1) \cup \mathsf{ap}(\mathsf{N}_2)$, $\mathsf{ap}((\nu x)\mathsf{N}) = \mathsf{ap}(\mathsf{N}) \setminus \{x\}$ and $\mathsf{ap}(\mathsf{0}) = \emptyset$.

CONVENTION 1 *In the remainder of this text, except where explicitly noted, we assume that expressions involving networks such as* $[\mathsf{P}]_A$ *are well-formed. In particular, quantifications like: "for all* P *and all* A, $[\mathsf{P}]_A$ *has property* X*" or even "for all* P, $[\mathsf{P}]_A$ *has property* X*" abbreviate the statement: "for all* P *and all* A *such that* $[\mathsf{P}]_A$ *is well-formed,* $[\mathsf{P}]_A$ *has property* X*".*

Access points have quite an influence on the behaviour of the calculus. For example $[\mathsf{0}]_\emptyset$ and $[\mathsf{0}]_x$ should never be equated by any semantically sound equivalence, although both contain an utterly inactive process. But they are different in that $[\mathsf{0}]_x$, unlike $[\mathsf{0}]_\emptyset$ can never be composed with another network that may input on x. $[\mathsf{0}]_x$ is a domain squatter! Messages on x will be routed to $[\mathsf{0}]_x$ but not to $[\mathsf{0}]_\emptyset$. For similar reasons we cannot allow $[\mathsf{0}]_A \equiv \mathsf{0}$, except when $A = \emptyset$.

5.3 Semantics

The semantics of π_{mlt} can be split into three parts. The semantics of processes, semantics of networks and the interplay between processes and networks.

5.3.1 Dynamics and Structural Congruence

Summarised in Figure 5.1 we find the semantics of π_{mlt}. The structural congruence is mostly a straightforward adaptation of the related rules in non-distributed π-calculi. The only axiom that may still require justification is

$$[(\nu x)\mathsf{P}]_A \;\equiv\; (\nu x)[\mathsf{P}]_{A\cup\{x\}}$$

The reason why scope extension must affect access points is that otherwise well-formedness of networks would not be closed under structural congruence. Of course we could use a less demanding rule, such as

$$x \in \mathsf{fin}(\mathsf{P}) \Rightarrow [(\nu x)\mathsf{P}]_A \;\equiv\; (\nu x)[\mathsf{P}]_{A\cup\{x\}}, \qquad x \notin \mathsf{fin}(\mathsf{P}) \Rightarrow [(\nu x)]\mathsf{P}_A \;\equiv\; (\nu x)[\mathsf{P}]_A.$$

(Here $\mathsf{fin}(\mathsf{P})$ returns all of P's free input subjects.) It rarely makes sense to restrict x in a local process P, without having x as an input subject, so we have opted for the syntactically simpler alternative.

Reductions of processes are integrated in the most liberal way possible: any reduction a process can do remains possible when located.

$$(\textsc{Intra}) \quad \frac{\mathsf{P} \to \mathsf{Q}}{[\mathsf{P}]_A \to [\mathsf{Q}]_A}$$

This is a key step towards π_{mlt} being incremental on top of π_t. Inter-site communication works by senders putting messages into the ether and receivers fetching them, as described by the next two rules.

$$(\textsc{Get}) \quad \frac{x \in A}{[\mathsf{P}]_A|\overline{x}\langle\vec{y}\rangle \to [\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle]_A} \qquad (\textsc{Send}) \quad \frac{x \notin A}{[\overline{x}\langle\vec{y}\rangle|\mathsf{P}]_A \to [\phi(\mathsf{P})]_A|\overline{x}\langle\vec{y}\rangle}$$

There is an asymmetry between (Get) and (Send). The latter ticks the local timer whenever a message leaves the location, but a message entering its target location

$$
\begin{array}{rcl@{\qquad}rcl}
\mathsf{fn}(\overline{x}\langle\vec{y}\rangle) & = & \{x,\vec{y}\} & \mathsf{bn}(\overline{x}\langle\vec{y}\rangle) & = & \emptyset \\
\mathsf{fn}([\mathsf{P}]_A) & = & \mathsf{fn}(\mathsf{P}) \cup A & \mathsf{bn}([\mathsf{P}]_A) & = & \mathsf{bn}(\mathsf{P}) \\
\mathsf{fn}(\mathsf{M} \mid \mathsf{N}) & = & \mathsf{fn}(\mathsf{M}) \cup \mathsf{fn}(\mathsf{N}) & \mathsf{bn}(\mathsf{M} \mid \mathsf{N}) & = & \mathsf{bn}(\mathsf{M}) \cup \mathsf{bn}(\mathsf{N}) \\
\mathsf{fn}((\nu x)\mathsf{N}) & = & \mathsf{fn}(\mathsf{N}) \setminus \{x\} & \mathsf{bn}((\nu x)\mathsf{N}) & = & \mathsf{bn}(\mathsf{N}) \cup \{x\} \\
\mathsf{fn}(0) & = & \emptyset & \mathsf{bn}(0) & = & \emptyset
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{M} \equiv_\alpha \mathsf{N} \Rightarrow \mathsf{M} \equiv \mathsf{N} & \mathsf{M} \mid \mathsf{N} \equiv \mathsf{N} \mid \mathsf{M} \\
\mathsf{L} \mid (\mathsf{M} \mid \mathsf{N}) \equiv (\mathsf{L} \mid \mathsf{M}) \mid \mathsf{N} & \mathsf{M} \mid 0 \equiv \mathsf{M} \\
x \notin \mathsf{fn}(\mathsf{M}) \Rightarrow \mathsf{M} \mid (\nu x)\mathsf{N} \equiv (\nu x)(\mathsf{M} \mid \mathsf{N}) & (\nu x)(\nu y)\mathsf{M} \equiv (\nu y)(\nu x)\mathsf{M} \\
(\nu x)0 \equiv 0 & [(\nu x)\mathsf{P}]_A \equiv (\nu x)[\mathsf{P}]_{A \cup \{x\}} \\
[0]_\emptyset \equiv 0 & \mathsf{P} \equiv \mathsf{Q} \Rightarrow [\mathsf{P}]_A \equiv [\mathsf{Q}]_A
\end{array}
$$

$$\text{(Intra)} \quad \frac{\mathsf{P} \to \mathsf{Q}}{[\mathsf{P}]_A \to [\mathsf{Q}]_A}$$

$$\text{(Send)} \quad \frac{x \notin A}{[\overline{x}\langle\vec{y}\rangle \mid \mathsf{P}]_A \to [\phi(\mathsf{P})]_A \mid \overline{x}\langle\vec{y}\rangle}$$

$$\text{(Get)} \quad \frac{x \in A}{[\mathsf{P}]_A \mid \overline{x}\langle\vec{y}\rangle \to [\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle]_A}$$

$$\text{(Loss)} \quad \frac{}{\overline{x}\langle\vec{y}\rangle \to 0}$$

$$\text{(Dupl)} \quad \frac{}{\overline{x}\langle\vec{y}\rangle \to \overline{x}\langle\vec{y}\rangle \mid \overline{x}\langle\vec{y}\rangle}$$

$$\text{(N-Par)} \quad \frac{\mathsf{M} \to \mathsf{M}'}{\mathsf{M} \mid \mathsf{N} \to \mathsf{M}' \mid \mathsf{N}}$$

$$\text{(N-Res)} \quad \frac{\mathsf{M} \to \mathsf{N}}{(\nu x)\mathsf{M} \to (\nu x)\mathsf{N}}$$

$$\text{(Cong)} \quad \frac{\mathsf{M} \equiv \mathsf{M}' \quad \mathsf{M}' \to \mathsf{N}' \quad \mathsf{N}' \equiv \mathsf{N}}{\mathsf{M} \to \mathsf{N}}$$

Figure 5.1: The inductive definition of the dynamics of π_{ml} networks, parametrised over a π-calculus, such as π_t of processes, with its associated dynamics, structural congruence and notions of free and bound names.

has not effects measurable by a timer. Is this the right choice for an asynchronous calculus? After all, ticking the clock allows a process to know if and when a message as left the location. Isn't that against the spirit of asynchrony? Why not use the following rule?

$$(\textsc{Send'}) \quad \frac{x \notin A}{[\overline{x}\langle\vec{y}\rangle|\mathsf{P}]_A \to [\mathsf{P}]_A|\overline{x}\langle\vec{y}\rangle}$$

Instead of (Get), why not use

$$(\textsc{Get'}) \quad \frac{x \in A}{[\mathsf{P}]_A|\overline{x}\langle\vec{y}\rangle \to [\phi(\mathsf{P}) \mid \overline{x}\langle\vec{y}\rangle]_A}?$$

Unfortunately, we cannot give a mathematical reason in support of our choice. We rely on aesthetics at this point in the development of π-calculi for distributed systems, because our intuition about the interplay between timing, location and failure is insufficient.

There is one argument for part of our preference though: modularity of reasoning. We want the equations holding for π_t-processes to stay valid when processes are located. This means that $\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}$ must imply $\mathcal{T}_{max} \vdash [\mathsf{P}]_A = [\mathsf{Q}]_A$. But if we were to replace (Send) by (Send'), so as to not tick timers when messages enter or leave a location then this requirement would be violated. To see why, define

$$\begin{aligned}
\mathsf{M} &= [x(v).x(w).\overline{w}]_x \\
\mathsf{P} &= (\nu y)(\mathsf{timer}^1(y.\overline{a}, 0) \,|\, \overline{x}\langle y\rangle \,|\, \overline{x}\langle y\rangle) \\
\mathsf{Q} &= (\nu y)(\mathsf{timer}^1(y.\overline{a}, 0) \,|\, \overline{x}\langle y\rangle \,|\, \mathsf{delay}^1(\overline{x}\langle y\rangle))
\end{aligned}$$

where a is a fresh name. It is easy to see that $\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}$, cf. Example 7. Unfortunately,

$$\mathsf{M} \,|\, [\mathsf{P}]_\emptyset \Downarrow_a \quad \text{while} \quad \mathsf{M} \,|\, [\mathsf{Q}]_\emptyset \not\Downarrow_a \,. \tag{5.1}$$

To see that $\mathsf{M} \,|\, [\mathsf{P}]_\emptyset \Downarrow_a$, consider

$$\begin{aligned}
\mathsf{M} \,|\, [\mathsf{P}]_\emptyset &\to \mathsf{M} \,|\, (\nu y)(\overline{x}\langle y\rangle \,|\, [\mathsf{timer}^1(y.\overline{a}, 0) \,|\, \overline{x}\langle y\rangle]_y) && \text{by (Send')} \\
&\to \mathsf{M} \,|\, (\nu y)(\overline{x}\langle y\rangle \,|\, \overline{x}\langle y\rangle \,|\, [\mathsf{timer}^1(y.\overline{a}, 0)]_y) && \text{by (Send')} \\
&\to (\nu y)([x(w).\overline{w}]_x \,|\, \overline{x}\langle y\rangle \,|\, [\mathsf{timer}^1(y.\overline{a}, 0)]_y) \\
&\to (\nu y)([\overline{y}]_x \,|\, [\mathsf{timer}^1(y.\overline{a}, 0)]_y) \\
&\to [0]_x \,|\, (\nu y)(\overline{y} \,|\, [\mathsf{timer}^1(y.\overline{a}, 0)]_y) \\
&\to [0]_x \,|\, (\nu y)[\overline{y} \,|\, \mathsf{timer}^1(y.\overline{a}, 0)]_y \\
&\to [0]_x \,|\, (\nu y)[\overline{a}]_y \\
&\to [0]_x \,|\, \overline{a} \,|\, (\nu y)[0]_y \downarrow_a
\end{aligned}$$

But the second output $\overline{x}\langle y\rangle$ is inside $\mathsf{delay}^1(\overline{x}\langle y\rangle)$ and can only be freed when times passes in $[\mathsf{Q}]_\emptyset$. That will inevitable time-out $\mathsf{timer}^1(y.\overline{a}, 0)$, removing a from the

free names of $\mathsf{timer}^1(y.\overline{a}, 0)$. As one can easily verify, this means that $\mathsf{M} \,|\, [\mathsf{Q}]_\emptyset \not\Downarrow_a$. A similar argument shows that having (GET') together with (SEND') is also not a good idea. Just define

$$\begin{aligned} \mathsf{P} &= (\nu y)(\mathsf{timer}^2(y.\overline{a}, 0) \,|\, \overline{x}\langle y\rangle \,|\, \overline{x}\langle y\rangle) \\ \mathsf{Q} &= (\nu y)(\mathsf{timer}^2(y.\overline{a}, 0) \,|\, \overline{x}\langle y\rangle \,|\, \mathsf{delay}^1(\overline{x}\langle y\rangle)) \end{aligned}$$

with M as before, then 5.1 still holds. So it is the lack of a Decomposition Theorem together with our required implication that rule out (SEND'), (GET) and (SEND'), (GET'). Of course one could turn the argument on its head and claim that the examples show the implication ($\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q} \Rightarrow \mathcal{T}_{max} \vdash [\mathsf{P}]_A = [\mathsf{Q}]_A$) to be unreasonable.

Unfortunately, we don't even have a contentious argument to prefer (SEND) over (SEND'), since $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ implies $\mathcal{T} \vdash \mathsf{P} \,|\, \overline{x}\langle\vec{y}\rangle = \mathsf{Q} \,|\, \overline{x}\langle\vec{y}\rangle$, but also $\mathcal{T} \vdash \phi(\mathsf{P}) \,|\, \overline{x}\langle\vec{y}\rangle = \phi(\mathsf{Q}) \,|\, \overline{x}\langle\vec{y}\rangle$. The reason for choosing (SEND) is mostly a guess. We expect the subsequent development in this field to suggest more compelling justifications. Here is a proposal for investigations.

CONJECTURE 2 The identity map on π_{mlt} processes is a fully abstract encoding between π_{mlt} with (SEND), (GET') and π_{mlt} with (SEND), (GET), using any reasonable equivalence that abstracts τ-actions.

Of the remaining reduction rules, those for message failure have already been discussed and the remaining ones are well known from conventional π-calculi. The most interesting remaining design choice is the absence of timestepping in (N-PAR). Why

$$(\text{N-PAR})\frac{\mathsf{M} \to \mathsf{M}'}{\mathsf{M}|\mathsf{N} \to \mathsf{M}'|\mathsf{N}}$$

rather than

$$(\text{N-PAR}')\frac{\mathsf{M} \to \mathsf{M}'}{\mathsf{M}|\mathsf{N} \to \mathsf{M}'|\phi(\mathsf{N})}$$

when the underlying process calculus has timers? How is time passing in different sites coordinated? The answer is "not at all"! This may be a surprising choice given that π_{mlt} is based on π_t. But in distributed systems clock-drift between sites is often considerable and several orders of magnitude about the duration of atomic computational steps. The easiest way to model such weak inter-site clock synchronisation is by not guaranteeing inter-site clock synchronisation at all. This design decision errs on the side of caution and but will have to be reevaluated later when more realistic models of distributed computation are required. After all, modern clock-synchronisation algorithms can reduce inter-site clock-drift below the average latency for remote communication [73] (which is still orders of magnitude above the duration of computational steps).

Example Reductions

Let's have a look at some more example reductions. Consider the network

$$[\overline{x}\langle y\rangle \mid \mathsf{timer}^3(z(v),0)]_z \mid [!x(v).\overline{z}\langle v\rangle]_x.$$

All it does is send a message on x to anther site where it gets forwarded to z and sent back to the originating site. There a timer is set that allows 2 units of to pass before it refuses to accept the forwarded message. A typical reduction sequence would be as follows.

$$\begin{aligned}
[\overline{x}\langle y\rangle \mid \mathsf{timer}^3(z(v),0)]_z \mid [!x(v).\overline{z}\langle v\rangle]_x &\rightarrow [\mathsf{timer}^2(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid [\overline{x}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid [\overline{z}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0) \mid \overline{z}\langle y\rangle]_z \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [0]_z \mid [!x(v).\overline{z}\langle v\rangle]_x
\end{aligned}$$

Nothing went wrong here. But message failures and idling can act non-deterministically and produce different reduction sequences. Here is an example.

$$\begin{aligned}
[\overline{x}\langle y\rangle \mid \mathsf{timer}^3(z(v),0)]_z \mid [!x(v).\overline{z}\langle v\rangle]_x &\rightarrow [\mathsf{timer}^2(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^2(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid [\overline{x}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid [\overline{z}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0) \mid \overline{z}\langle y\rangle]_z \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0) \mid \overline{x}\langle y\rangle]_z \mid [\overline{x}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0) \mid \overline{x}\langle y\rangle]_z \mid [\overline{z}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0) \mid \overline{x}\langle y\rangle]_z \mid [\overline{z}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0) \mid \overline{x}\langle y\rangle]_z \mid \overline{z}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [0]_z \mid \overline{z}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\overline{z}\langle y\rangle]_z \mid [!x(v).\overline{z}\langle v\rangle]_x
\end{aligned}$$

The last reduction sequence shows one crucial problem with setting timers to detect message loss: time-out may be caused not because of message loss but because

messages arrive too late.

$$\begin{aligned}
[\overline{x}\langle y\rangle \mid \mathsf{timer}^3(z(v),0)]_z \mid [!x(v).\overline{z}\langle v\rangle]_x &\rightarrow [\mathsf{timer}^2(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{timer}^1(z(v),0)]_z \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{0}]_z \mid \overline{x}\langle y\rangle \mid [!x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{0}]_z \mid [\overline{x}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\mathsf{0}]_z \mid [\overline{z}\langle y\rangle \mid !x(v).\overline{z}\langle v\rangle]_x \\
&\rightarrow [\overline{z}\langle y\rangle]_z \mid [!x(v).\overline{z}\langle v\rangle]_x
\end{aligned}$$

What these examples ought to communicate is that the new calculus works very much like π_t except for the addition of sites which server two functions: localise time-passing and structure message passing into parts that are failure free and parts that are not.

Free Input and Output Names

The notions of *free input names* and *free output names* have a straightforward extension to networks.

$$\begin{array}{lcllcl}
\mathsf{fin}(\overline{x}\langle\vec{y}\rangle) & = & \emptyset & \mathsf{fon}(\overline{x}\langle\vec{y}\rangle) & = & \{x\} \\
\mathsf{fin}([\mathsf{P}]_A) & = & \mathsf{fin}(\mathsf{P}) & \mathsf{fon}([\mathsf{P}]_A) & = & \mathsf{fon}(\mathsf{P}) \\
\mathsf{fin}(\mathsf{M}|\mathsf{N}) & = & \mathsf{fin}(\mathsf{M}) \cup \mathsf{fin}(\mathsf{N}) & \mathsf{fon}(\mathsf{M}|\mathsf{N}) & = & \mathsf{fon}(\mathsf{M}) \cup \mathsf{fon}(\mathsf{N}) \\
\mathsf{fin}((\nu x)\mathsf{N}) & = & \mathsf{fin}(\mathsf{N}) \setminus \{x\} & \mathsf{fon}((\nu x)\mathsf{N}) & = & \mathsf{fon}(\mathsf{N}) \setminus \{x\} \\
\mathsf{fin}(\mathsf{0}) & = & \emptyset & \mathsf{fon}(\mathsf{0}) & = & \emptyset
\end{array}$$

Active Names and Insensitivity

Similarly, active names and insensitivity must be extended to networks.

$$\begin{aligned}
\mathsf{an}(\overline{x}\langle\vec{y}\rangle) &= \{x\} \\
\mathsf{an}([\mathsf{P}]_A) &= \mathsf{an}(\mathsf{P}) \\
\mathsf{an}(\mathsf{M}|\mathsf{N}) &= \mathsf{an}(\mathsf{M}) \cup \mathsf{an}(\mathsf{N}) \\
\mathsf{an}((\nu x)\mathsf{N}) &= \mathsf{an}(\mathsf{N}) \setminus \{x\} \\
\mathsf{an}(\mathsf{0}) &= \emptyset
\end{aligned}$$

Definition 56 A network M is *insensitive* if for all reduction sequences $\mathsf{M} \rightarrow\!\!\!\!\rightarrow \mathsf{N}$ it is the case that $\mathsf{an}(\mathsf{N}) = \emptyset$. The set of all insensitive networks is denoted ins.

Contexts

DEFINITION 57 *Network contexts* are given by the grammar

$$C[\cdot] ::= [\cdot] \quad | \quad C[\cdot] \,|\, \mathsf{N} \quad | \quad (\nu x)C[\cdot]$$

DEFINITION 58 If $C[\cdot]$ and $C'[\cdot]$ are network contexts, then $C[\cdot]$ *generically reduces to* $C'[\cdot]$, denoted $C[\cdot] \twoheadrightarrow C'[\cdot]$ if for all N some reduction sequence $C[\mathsf{N}] \twoheadrightarrow C'[\mathsf{N}]$ exists.

Barbs

The barbs for processes are those of the underlying process calculus. For networks we have

$$\frac{\mathsf{M}\downarrow_x \quad x \notin \mathsf{ap}(\mathsf{N})}{\mathsf{M}|\mathsf{N}\downarrow_x} \qquad \frac{\mathsf{M}\downarrow_x \quad x \neq a}{(\nu a)\mathsf{M}\downarrow_x} \qquad \frac{}{\overline{x}\langle\vec{y}\rangle\downarrow_x}$$

This choice of barb-forming rules deserves some justification. Why have we not included

$$\frac{\mathsf{P}\downarrow_x \quad x \notin A}{[\mathsf{P}]_A\downarrow_x}?$$

The answer is that we believe observers to be other processes. The above rule then follows, because our reduction rules make it impossible for other processes to immediately observe an output at x in, say, $[\overline{x}\langle\vec{y}\rangle \,|\, \mathsf{P}]_A, x \notin A$, despite $\overline{x}\langle\vec{y}\rangle \,|\, \mathsf{P}\downarrow_x$. One may reply here that we could relax the requirement about what entities may observe processes. It makes sense to allow the message passing fabric to also be an observer. Prima facie, this is a reasonable objection and the right way to resolve the issue would be to develop the theory of *strong* sound theories for π_{mlt}, along the lines of [61] and see what barbs it induces, but we will not do this here because we are really after sound theories.

The motivation behind requiring $x \notin \mathsf{ap}(\mathsf{N})$ in the leftmost rule is to prevent the observation of messages that are destined for locations that cannot possibly be part of an observer. Although it might ultimately more realistic to allow misrouting of messages, the present model simplifies things by prohibiting this possibility.

LEMMA 41 *1. If* $\mathsf{M} \equiv \mathsf{N}$ *and* $\mathsf{M}\downarrow_x$ *then* $\mathsf{N}\downarrow_x$.

2. If $\mathsf{M} \equiv \mathsf{N}$ *and* $\mathsf{M}\Downarrow_x$ *then* $\mathsf{N}\Downarrow_x$.

PROOF: By straightforward nested inductions. □

Barbed Congruences

In the previous chapter we asked if adding timers requires a new notion of barb. The answer was no. A similar question arises for π_{mlt}. Again the answer will turn out

to be a qualified "no". No, because our barbs will still just look for active output, but only in the ether. Outputs active within a site will not count as immediate observations. We will show that this choice is semantically correct, as we have done for π_t.

DEFINITION 59 A binary relation $\mathcal{R}$ on networks is a *π_{mlt}-congruence* if it it an equivalence, if $\equiv \subseteq \mathcal{R}$ and if $\mathsf{M}\ \mathcal{R}\ \mathsf{N}$ implies $C[\mathsf{M}]\ \mathcal{R}\ C[\mathsf{N}]$ for all network-contexts $C[\cdot]$.

DEFINITION 60 A symmetric binary relation $\mathcal{R}$ on networks is a *strong barbed bisimulation* if it is a π_{mlt}-congruence and if $\mathsf{M}\ \mathcal{R}\ \mathsf{N}$ implies the following.

- For all names x: $\mathsf{M}\downarrow_x$ implies $\mathsf{N}\downarrow_x$.
- Whenever $\mathsf{M}\rightarrow\mathsf{M}'$ then there is a network N' such that $\mathsf{N}\rightarrow\mathsf{N}'$ and $\mathsf{M}'\ \mathcal{R}\ \mathsf{N}'$.

It is easy to see that strong barbed bisimulations are closed under arbitrary unions. The largest strong barbed bisimulation is called *strong reduction congruence*. We denote it by $\overset{rc}{\sim}$.

The corresponding notions of *barbed bisimulation* and *reduction congruence* $\overset{rc}{\approx}$ are derived as explained in Chapter 2, by replacing $\downarrow_x$ with $\Downarrow_x$ and $\rightarrow$ with $\twoheadrightarrow$. Clearly, each strong barbed bisimulation is a barbed bisimulation.

Before continuing with the transitional semantics, we present examples of a (strong) barbed bisimilarity and its absence, which will play a role later.

EXAMPLE 9 For all network M:

$$\mathsf{M} \overset{rc}{\sim} \mathsf{M} \mid [(\nu x)\overline{x}]_{\emptyset}.$$

To see that this is the case, define

$$\mathcal{R} = \{(C[\mathsf{M}], C[\mathsf{M} \mid [(\nu x)\overline{x}]_{\emptyset}]) \mid \mathsf{M} \text{ is a network}, C[\cdot] \text{ is a context}\} \cup \equiv$$

It is easy to see that $\mathcal{R}$ is a barbed bisimulation.

EXAMPLE 10 If $a \neq x$ then

$$[\overline{x}]_{\emptyset} \overset{rc}{\approx} [(\nu a)(\overline{a} \mid a.\overline{x})]_{\emptyset} \qquad \text{but} \qquad [\overline{x}]_{\emptyset} \overset{rc}{\not\sim} [(\nu a)(\overline{a} \mid a.\overline{x})]_{\emptyset}.$$

but the processes are not equated by $\overset{rc}{\sim}$.

$$\text{(Intra)}\quad \frac{\mathsf{P} \xrightarrow{\tau} \mathsf{Q}}{[\mathsf{P}]_A \xrightarrow{\tau} [\mathsf{Q}]_A}$$

$$\text{(N-Out)}\quad \frac{\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{Q} \quad x \notin A \quad \{\vec{y}\} \cap A = \emptyset}{[\mathsf{P}]_A \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} [\mathsf{Q}]_{A\cup\{\vec{y}\}}}$$

$$\text{(N-In)}\quad \frac{\mathsf{P} \xrightarrow{x(\vec{z})} \mathsf{Q}}{[\mathsf{P}]_A \xrightarrow{x(\vec{z})} [\mathsf{Q}]_A}$$

$$\text{(EtherIn)}\quad \frac{}{\mathsf{0} \xrightarrow{x(\vec{z})}_n \overline{x}\langle\vec{z}\rangle}$$

$$\text{(EtherOut)}\quad \frac{}{\overline{x}\langle\vec{z}\rangle \xrightarrow{\overline{x}\langle\vec{z}\rangle}_n \mathsf{0}}$$

$$\text{(Loss)}\quad \frac{}{\overline{x}\langle\vec{z}\rangle \xrightarrow{\tau}_n \mathsf{0}}$$

$$\text{(Dupl)}\quad \frac{}{\overline{x}\langle\vec{z}\rangle \xrightarrow{\tau}_n \overline{x}\langle\vec{z}\rangle \mid \overline{x}\langle\vec{z}\rangle}$$

$$\text{(N-Par)}\quad \frac{\mathsf{M} \xrightarrow{l}_t \mathsf{M}' \quad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{N}) = \emptyset \quad t \in \{n, \epsilon\} \quad l = \overline{x}\langle(\nu\vec{y})\vec{z}\rangle \Rightarrow x \notin \mathsf{ap}(\mathsf{N})}{\mathsf{M}|\mathsf{N} \xrightarrow{l}_t \mathsf{M}'|\mathsf{N}}$$

$$\text{(N-Com)}\quad \frac{\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_s \mathsf{M}' \quad \mathsf{N} \xrightarrow{x(\vec{z})}_t \mathsf{N}' \quad \{\vec{y}\} \cap \mathsf{fn}(\mathsf{N}) = \emptyset \quad s, t \in \{n, \epsilon\}, s \neq t}{\mathsf{M}|\mathsf{N} \xrightarrow{\tau} (\nu\vec{y})(\mathsf{M}'|\mathsf{N}')}$$

$$\text{(N-Res)}\quad \frac{\mathsf{M} \xrightarrow{l}_t \mathsf{N} \quad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l) \quad t \in \{n, \epsilon\}}{(\nu x)\mathsf{M} \xrightarrow{l}_t (\nu x)\mathsf{N}}$$

$$\text{(N-Open)}\quad \frac{\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_t \mathsf{N} \quad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\} \quad t \in \{n, \epsilon\}}{(\nu v)\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle}_t \mathsf{N}}$$

$$\text{(N-Alpha)}\quad \frac{\mathsf{M} \equiv_\alpha \mathsf{M}' \quad \mathsf{M}' \xrightarrow{l}_t \mathsf{N}' \quad \mathsf{N} \equiv_\alpha \mathsf{N}' \quad t \in \{n, \epsilon\}}{\mathsf{M} \xrightarrow{l}_t \mathsf{N}}$$

Figure 5.2: A synchronous transition system for the asynchronous timed π-calculus with locations and message loss. Here $\rightarrow_\epsilon$ stands for $\rightarrow$.

5.3.2 Transitional Semantics

Figure 5.2 presents a labelled transition system that captures the same computations as the reductions in Figure 5.1. We hesitate to call it synchronous because of the way the ether generates observations. The ether can never refuse to accept a message from a site. It is of infinite capacity. This is modelled by the following rules

$$(\textsc{EtherIn}) \quad \frac{}{\mathsf{0} \xrightarrow{x(\vec{z})}_n \overline{x}\langle\vec{z}\rangle} \qquad (\textsc{EtherOut}) \quad \frac{}{\overline{x}\langle\vec{z}\rangle \xrightarrow{\overline{x}\langle\vec{z}\rangle}_n \mathsf{0}}$$

which are very much like to corresponding asynchronous rules in the asynchronous π-calculus, cf. §2.2.1. It is not clear how to have entirely synchronous semantics. The remaining rules are straightforward adaptations of the reduction rules.

5.4 Equivalences

With the basic definitions out of the way, it is time to study the crucial properties of the calculus, to see if our constructions make sense at all. As with π_t, the investigations centre around the maximum sound theory. But the calculus is now a 2-sorted algebra, so there is more than one way of defining the notions of sound theory. We show that this is no reason to worry because all of them induce the same unique maximum theory. With the problem of the right definition out of the way, we then verify that the definition of barb in §5.3.1 is semantically correct because the induced reduction congruence coincides with the maximum sound theory. Finally, we propose another labelled transition system with an associated notion of bisimulation for tractable compositional reasoning. Unfortunately, this leads only to a sound approximation of the maximum sound theory but a characterisation is not in sight. Timers not being synchronised across sites precludes applications of the technology developed for π_t's characterisation theorem. It seems π_{mlt} is more like an untimed pi-calculus rather than π_t.

5.4.1 The Maximum Sound Theory for π_{mlt}

The overall architecture of this section is much like that of §4.4.1 and hence [61]. The proofs are somewhat more involved due to the possibility of message failure but not much.

Basic Definitions and Facts

All concepts fixed in Definition 22 remain unchanged, except for the notion of π_{mlt}-logic which must take account of access points.

Definition 61 A *π_{mlt}-logic* is a logic where formulae are pairs (M, N) of networks such that $\mathsf{ap}(\mathsf{M}) = \mathsf{ap}(\mathsf{N})$.

Without the coincidence of access points for related networks, no π_{mlt}-logic could be a congruence, as explained in §5.2.
But what is a π_{mlt}-theory? π_t is a 1-sorted algebra, but π_{mlt} isn't. How should π_{mlt}-theories account for equalities of the underlying processes? Our design goals suggest that

$$\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q} \quad \Rightarrow \quad \mathcal{T}_{max} \vdash [\mathsf{P}]_A = [\mathsf{Q}]_A \tag{5.2}$$

ought to hold. In this implication, $\mathcal{T}_{max}$ in the premise is the maximum sound theory for π_t, as described in the previous chapter, while in the conclusion, $\mathcal{T}_{max}$ refers to the maximum sound theory on networks, whose existence this section sets out to prove. We will usually not distinguish syntactically between these relations, hoping that the reader can always find enough contextual clues for disambiguation.

There seem to be several ways of coherently defining π_{mlt}-theories satisfying 5.2.

- One could treat π_{mlt} as a 1-sorted algebra, with $[\mathsf{P}]_A$ being a nullary operation for all P, A. Then π_{mlt}-theories are defined like π_t-theories just over networks instead of processes and 5.2 is proven as a theorem (hopefully, because a failure to derive 5.2 would indicate problems with the whole approach).

- Alternatively, we could proceed as in the previous suggestion, except that 5.2 is enforced by definition, via an additional axiom in the requirements of what it means for $\mathcal{T}$ to be a π_{mlt}-theory:

$$\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q} \Rightarrow \mathcal{T} \vdash [\mathsf{P}]_A = [\mathsf{Q}]_A \tag{5.3}$$

These two proposals decree processes to be second class citizens. This is not necessary

- We could also define the notion of π_t-theory and π_{mlt}-theory simultaneously, as a pair $(\mathcal{T}, \mathcal{T}')$, where $\mathcal{T}$ relates networks and $\mathcal{T}'$ processes. The two are connected by the following axiom.

$$\mathcal{T}' \vdash \mathsf{P} = \mathsf{Q} \Rightarrow [\mathsf{P}]_A = [\mathsf{Q}]_A.$$

Some variations of these proposals are possible, but we restrict our efforts to showing that these three all result in the same congruence.
We cannot require $\mathcal{T} \vdash [\mathsf{P}]_A = [\mathsf{Q}]_A$ to imply $\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}$, for $[\overline{x}]_{\{x,y\}}$ and $[\overline{y}]_{\{x,y\}}$ are both unable to interact in any form or shape, yet $\mathcal{T}_{max} \nvdash \overline{x} = \overline{y}$.

DEFINITION 62 A *1-theory* is a consistent and reduction-closed π_{mlt}-logic $(\mathcal{T}, \vdash)$ where entailment is defined inductively by the constrains given next.

- $(\mathsf{M}, \mathsf{N}) \in \mathcal{T}$ implies $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$.

- $\mathsf{M} \equiv \mathsf{N}$ implies $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$.
- $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ implies $\mathcal{T} \vdash \mathsf{N} = \mathsf{M}$.
- $\mathcal{T} \vdash \mathsf{L} = \mathsf{M}$ and $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ imply $\mathcal{T} \vdash \mathsf{L} = \mathsf{N}$.
- $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ implies $\mathcal{T} \vdash C[\mathsf{M}] = C[\mathsf{N}]$ for all network contexts $C[\cdot]$.

A *2-theory* is a consistent and reduction-closed π_{mlt}-logic $(\mathcal{T}, \vdash)$ where entailment is defined by the rules for 1-theories with the additional requirement that

- $\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}$ implies $\mathcal{T} \vdash [\mathsf{P}]_A = [\mathsf{Q}]_A$ for all appropriate A.

A *3-theory* is a pair $(\mathcal{T}, \mathcal{T}')$ such that $\mathcal{T}'$ is a consistent and reduction-closed π_t-theory while $(\mathcal{T}, \vdash)$ is a consistent and reduction-closed π_{mlt}-logic where entailment is defined by the rules for 1-theories with the additional requirement that

- $\mathcal{T}' \vdash \mathsf{P} = \mathsf{Q}$ implies $\mathcal{T} \vdash [\mathsf{P}]_A = [\mathsf{Q}]_A$ for all appropriate A.

1-theories and 2-theories are *sound* if the identify all insensitive networks. A 3-theory $(\mathcal{T}, \mathcal{T}')$ is *sound* if $\mathcal{T}$ identifies all insensitive networks and $\mathcal{T}'$ identifies all insensitive π_t-processes.

EXAMPLE 11 Barbed Congruence $\overset{rc}{\approx}$ is a sound 1-theory: reduction-closure is by definition. Consistency follows because barbed bisimulations preserve weak barbs. To see that $\overset{rc}{\approx}$ identifies all insensitive terms, consider

$$\mathcal{R} = \{(C[\mathsf{M}], C[\mathsf{N}]) \mid C[\cdot] \text{ context}, \mathsf{M}, \mathsf{N} \text{ insensitive}\} \cup \equiv .$$

It is easy to check that $\mathcal{R}$ is a barbed congruence, as required.

The aim of this section is to verify the next theorem, which states that 1, 2 and 3-theories all give raise to the same notion of equivalence.

THEOREM 27 *1. There exists a unique maximum sound 1-theory $\mathcal{T}^1_{max}$, but there is no maximum 1-theory.*

2. There exists a unique maximum sound 2-theory $\mathcal{T}^2_{max}$, but there is no maximum 2-theory.

3. There exists a unique maximum sound 3-theory $\mathcal{T}^3_{max} = (\mathcal{T}_3, \mathcal{T}'_3)$, but there is no maximum 3-theory.

4. $\mathcal{T}^1_{max} = \mathcal{T}^2_{max} = \mathcal{T}_3$. In addition, $\mathcal{T}'_3$ is the maximum sound π_t-theory.

PROOF: We prove (4) from (1, 2, 3), which we establish later.

Clearly, if $\mathcal{T}$ is a sound 2-theory, then $\mathcal{T}, \mathcal{T}_{max}$ is a sound 3-theory. Similarly, if $(\mathcal{T}^n, \mathcal{T}^t)$ is a 3-theory, then $\mathcal{T}^n$ is a 1-theory. But then, if $\mathcal{T}^i_{max}$ is was the maximum i-theory $(i = 1, 2, 3)$ then

$$\mathcal{T}^2_{max} \subseteq \mathcal{T}^{3,n}_{max} \subseteq \mathcal{T}^1_{max},$$

where $\mathcal{T}^3_{max} = (\mathcal{T}^{3,n}_{max}, \mathcal{T}^{3,t}_{max})$. We will now show that $\mathcal{T}^1_{max}$ is a 2-theory. This means that

$$\mathcal{T}^2_{max} = \mathcal{T}^{3,n}_{max} = \mathcal{T}^1_{max},$$

provided these maximum i-theories exist. Define

$$\mathcal{T} \quad = \quad \{([\mathsf{P}]_A, [\mathsf{Q}]_A) \mid \mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}\} \cup \{(\mathsf{M}, \mathsf{N}) \mid \mathsf{M}, \mathsf{N} \text{ insensitive}\}.$$

We show that $\mathcal{T}$ is a sound 2-theory. The only non-trivial facts to check are reduction-closure and consistency. For the former, we show by induction on the derivation of $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ that for all networks L: whenever $\mathsf{M}|\mathsf{L} \to \mathsf{M}'$ there there is a reduction sequence $\mathsf{N}|\mathsf{L} \to \mathsf{N}'$ with $\mathcal{T} \vdash \mathsf{M}' = \mathsf{N}'$.

$\underline{\mathsf{M} = [\mathsf{P}]_A, \mathsf{N} = [\mathsf{Q}]_A, \mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}.}$ Induction on the derivation of $\mathsf{M}|\mathsf{L} \to \mathsf{M}'$.

- $\underline{[\mathsf{P}]_A|\mathsf{L} \to [\mathsf{P}']_A|\mathsf{L}.}$ Then strong reduction-closure of $\mathcal{T}_{max}$ delivers a matching reduction $[\mathsf{P}]_A \to [\mathsf{P}']_A$.
- $\underline{[\mathsf{P}]_A|\mathsf{L} \to [\mathsf{P}]_A|\mathsf{L}'.}$ Then clearly $[\mathsf{Q}]_A|\mathsf{L} \to [\mathsf{Q}]_A|\mathsf{L}'$ matches.
- $\underline{[\mathsf{P}]_A|\overline{x}\langle\vec{y}\rangle|\mathsf{L}' \to [\mathsf{P}|\overline{x}\langle\vec{y}\rangle]_A|\mathsf{L}'.}$ Then $x \in A$ and $[\mathsf{Q}]_A|\overline{x}\langle\vec{y}\rangle|\mathsf{L}' \to [\mathsf{Q}|\overline{x}\langle\vec{y}\rangle]_A|\mathsf{L}'$. This is a match by congruency of $\mathcal{T}_{max}$.
- $\underline{[(\nu\vec{y})(\mathsf{P}'|\overline{x}\langle\vec{z}\rangle)] \mid \mathsf{L} \to (\nu\vec{y})([\phi(\mathsf{P}')]_{A\cup\{\vec{y}\}} \mid \overline{x}\langle\vec{z}\rangle) \mid \mathsf{L}.}$ Then $\{\vec{y}\} \subseteq \{\vec{z}\} \setminus \{x\}$. From Corollary 1 we know that $\mathsf{Q} \equiv (\nu\vec{y})(\mathsf{Q}'|\overline{x}\langle\vec{z}\rangle)$ where $\mathcal{T}_{max} \vdash \phi(\mathsf{P}') = \phi(\mathsf{Q}')$. Hence $[\mathsf{Q}]_A \mid \mathsf{L} \to (\nu\vec{y})([\phi(\mathsf{Q}')]_{A\cup\{\vec{y}\}} \mid \overline{x}\langle\vec{z}\rangle) \mid \mathsf{L}$ is a matching reduction.

$\underline{\mathcal{T} \vdash \mathsf{M}'' = \mathsf{N}'', \mathsf{M} = C[\mathsf{M}''], \mathsf{N} = C[\mathsf{N}''].}$ We proceed by induction on the derivation of $C[\mathsf{M}''] \mid \mathsf{L} \to \mathsf{M}'$.

- $\underline{C[\mathsf{M}''] \mid \mathsf{L} \to \mathsf{M}''' \mid \mathsf{L}.}$ Use the (IH).
- $\underline{C[\mathsf{M}''] \mid \mathsf{L} \to C[\mathsf{M}''] \mid \mathsf{L}'.}$ The matching reduction sequence is $C[\mathsf{N}''] \mid \mathsf{L} \to\!\!\!\!\to C[\mathsf{N}''] \mid \mathsf{L}'$ by congruency.
- $\underline{C[\mathsf{M}''] \mid \mathsf{L} \to (\nu\vec{x})(C'[\mathsf{M}''] \mid \mathsf{L}').}$ Then $C[\mathsf{N}''] \mid \mathsf{L} \to (\nu\vec{x})(C'[\mathsf{N}''] \mid \mathsf{L}')$ is a match.
- $\underline{C[\mathsf{M}''] \mid \mathsf{L} \to (\nu\vec{x})(C[\mathsf{M}'''] \mid \mathsf{L}').}$ Now induction on the structure of $C[\cdot]$.
  - $\underline{C[\cdot] = [\cdot].}$ The matching sequence is constructed using the outermost (IH).

$\underline{C[\cdot] = C'[\cdot]|\mathsf{L}'.}$ We use the innermost (IH).

$\underline{C[\cdot] = (\nu x)C'[\cdot].}$ Then we must match $(\nu x)(C'[\mathsf{M}''] \mid \mathsf{L}) \rightarrow \mathsf{M}'$. By an easy induction of this reduction we know that $C'[\mathsf{M}''] \mid \mathsf{L} \rightarrow \mathsf{M}'''$. where $\mathsf{M}' \equiv (\nu x)\mathsf{M}'''$. By the inner (IH) there must be a reduction sequence $C'[\mathsf{N}''] \mid \mathsf{L} \twoheadrightarrow \mathsf{N}'''$ with $\mathcal{T} \vdash \mathsf{M}''' = \mathsf{N}'''$, hence $C[\mathsf{N}''] \mid \mathsf{L} \twoheadrightarrow (\nu x)\mathsf{N}'''$ is an appropriate reduction sequence.

The remaining cases are immediate. For consistency, we show by induction on the derivation of $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ that $\mathsf{M}\Downarrow_x$ implies $\mathsf{N}\Downarrow_x$. This induction is straightforward. The only slightly non-trivial case is $\mathcal{T} \vdash [\mathsf{P}]_A = [\mathsf{Q}]_A$ because $\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}$. If $[\mathsf{P}]_A \Downarrow_x$ then (Lemma 41.2) $\mathsf{P}\Downarrow_x$ and $x \notin A$. But $\mathcal{T}_{max}$, too, preserves weak barbs (Lemma 13), so $\mathsf{Q}\Downarrow_x$ and $[\mathsf{Q}]_A \Downarrow_x$.

An immediate consequence of weak barb preservation is consistency for $\mathsf{0}\not\Downarrow_x$ but $\overline{x}\langle\vec{y}\rangle\Downarrow_x$. □

The next task is to conclude the proof of the theorem by verifying that the maximum sound i-theories do exist. In this section, $\mathcal{T}$ ranges over 3-theories.

LEMMA 42 *Let $\mathcal{T}$ be an i-theory ($i = 1, 2, 3$).*

1. *$\mathcal{T}$ is reduction-closed if and only if,*
   - *whenever $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$, then, for all network contexts $C[\cdot]$, $C[\mathsf{M}] \twoheadrightarrow \mathsf{M}'$ implies $C[\mathsf{N}] \twoheadrightarrow \mathsf{N}'$, for some N' with $\mathcal{T} \vdash \mathsf{M}' = \mathsf{N}'$.*
   - *In addition, if $i = 3$, whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$, then, for all contexts $C[\cdot]$, $C[\mathsf{P}] \twoheadrightarrow \mathsf{P}'$ implies $C[\mathsf{Q}] \twoheadrightarrow \mathsf{Q}'$, for some Q' with $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$.*
2. *$\mathcal{T}$ is reduction-closed if and only if,*
   - *whenever $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$, then, for all network contexts $C[\cdot]$, $C[\mathsf{M}] \twoheadrightarrow \mathsf{M}'$ implies $C[\mathsf{N}] \twoheadrightarrow \mathsf{N}'$, for some N' with $\mathcal{T} \vdash \mathsf{M}' = \mathsf{N}'$.*
   - *In addition, if $i = 3$, whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$, then, for all contexts $C[\cdot]$, $C[\mathsf{P}] \twoheadrightarrow \mathsf{P}'$ implies $C[\mathsf{Q}] \rightarrow \mathsf{Q}'$, for some Q' with $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$.*
3. *Let $\mathcal{T} = \Sigma_{i\in I}\mathcal{T}_i$. $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ if and only if there are $i_1, ..., i_n \in I$ such that*

$$\mathcal{T}_{i_1} \vdash \mathsf{M}_0 = \mathsf{M}_1,\ ...\mathcal{T}_{i_{n-1}} \vdash \mathsf{M}_{n-2} = \mathsf{M}_{n-1},\ \mathcal{T}_{i_n} \vdash \mathsf{M}_{n-1} = \mathsf{M}_n,$$

   where $\mathsf{M} = \mathsf{M}_0$ and $\mathsf{M}_n = \mathsf{N}$.
4. *If $\mathcal{T}_i$ is reduction-closed for all $i \in I$, then so is $\Sigma_{i\in I}\mathcal{T}_i$.*
5. *$|\,\mathcal{T}_i\,| \subseteq |\Sigma_{i\in I}\,\mathcal{T}_i\,|$ for all i.*

PROOF: For (1) we use induction on the structure of the relevant contexts to show ($\Rightarrow$). The converse direction is trivial. (2) is by (1) and Lemma 10.2. (3, 4, 5) are proven as their counterparts in Lemma 10. $\Box$

DEFINITION 63 Networks M and N are *i-incompatible* ($i = 1, 2, 3$), written $\mathsf{M} \#_i \mathsf{N}$ if for all sound i-theories $\mathcal{T}$: $\mathcal{T} \not\vdash \mathsf{M} = \mathsf{N}$.

LEMMA 43 *Let $i = 1, 2, 3$.*

1. *Assume that $\mathcal{T}$ is a sound i-theory with $\mathcal{T} \vdash (\nu\vec{x})[\mathsf{P}]_A = \mathsf{0}$ for all P, some A and some $\vec{x}$ such that $\{\vec{x}\} \cap \mathsf{fn}(\mathsf{P}) = \emptyset$. Then $\mathcal{T}$ is inconsistent.*
2. *If $\mathsf{M}\Downarrow_x$ but $\mathsf{N}\not\Downarrow_x$ then $\mathsf{M} \#_i \mathsf{N}$.*
3. *Let $\mathcal{T}$ be a sound i-theory If $\mathcal{T} \vdash P = Q$, then $\mathsf{M}\Downarrow_x$ if and only if $\mathsf{N}\Downarrow_x$.*
4. *If $C[\cdot]$ is a network context and $C[\mathsf{M}]$ # $C[\mathsf{N}]$ then $\mathsf{M} \ \#_i \ \mathsf{N}$.*
5. *$\mathsf{P} \#_i \mathsf{Q}$ does not imply $[\mathsf{P}]_A \#_i [\mathsf{Q}]_A$.*

PROOF: For (1) we show $\mathcal{T} \vdash \mathsf{0} = \mathsf{M}$ for all M by induction on the structure of M. Assume $\mathsf{M} = \mathsf{N}_1|\mathsf{N}_2$, then by (IH) $\mathcal{T} \vdash \mathsf{0} = \mathsf{N}_i$, hence $\mathcal{T} \vdash \mathsf{0}|\mathsf{0} = \mathsf{M}$. But $\mathcal{T} \vdash \mathsf{0} = \mathsf{0}|\mathsf{0}$. Next, let $\mathsf{M} = \overline{a}\langle\vec{b}\rangle$. By assumption $\mathcal{T} \vdash \mathsf{0} = (\nu\vec{x})[\overline{a}\langle\vec{b}\rangle]_A$. But $[\overline{a}\langle\vec{b}\rangle]_A \rightarrow \overline{a}\langle\vec{b}\rangle|(\nu\vec{x})[\mathsf{0}]_A$. As $(\nu\vec{x})[\mathsf{0}]_A$ is insensitive and $\mathsf{0} \twoheadrightarrow \mathsf{0}$ is the only available reduction sequence, we use reduction-closure to conclude to $\mathcal{T} \vdash \mathsf{0} = \overline{a}\langle\vec{b}\rangle$. The remaining cases are immediate from the (IH).

For (2) assume $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ and choose an arbitrary process P. Define

$$C[\cdot] = (\nu z)([z.\mathsf{P}]_A \mid (\nu\vec{y})([\cdot] \mid [x(\vec{v}).\overline{z}]_{\{x\}}))$$

where z is a fresh name, $\mathsf{fn}(\mathsf{M}) \cup \mathsf{fn}(\mathsf{N}) = \{\vec{y}\}$ and A is a set of names such that $[z.\mathsf{P}]_A$ is well-formed. Now assume $\mathsf{M}\Downarrow_x$ because $\mathsf{M} \twoheadrightarrow (\nu\vec{a})(\mathsf{M}'|\overline{x}\langle\vec{z}\rangle)$, where $x \notin \{\vec{a}\}$. Then

$$\begin{aligned}
C[\mathsf{M}] &\rightarrow C[\mathsf{M}'] \\
&\rightarrow (\nu z)([z.\mathsf{P}]_A \mid (\nu\vec{a}\vec{y}x)(\mathsf{M}'' \mid \overline{x}\langle\vec{z}\rangle \mid [x(\vec{v}).\overline{z}]_{\{x\}})) \\
&\rightarrow (\nu z)([z.\mathsf{P}]_A \mid (\nu\vec{a}\vec{y}x)(\mathsf{M}'' \mid [\overline{z}]_{\{x\}})) \\
&\rightarrow (\nu z)([z.\mathsf{P}]_A \mid (\nu\vec{a}\vec{y}x)(\mathsf{M}'' \mid [\mathsf{0}]_{\{x\}} \mid \overline{z})) \\
&\rightarrow (\nu z)[\overline{z} \mid z.\mathsf{P}]_A \mid (\nu\vec{a}\vec{y}x)(\mathsf{M}'' \mid [\mathsf{0}]_{\{x\}}) \\
&\rightarrow (\nu z)[\mathsf{P}]_A \mid (\nu\vec{a}\vec{y}x)(\mathsf{M}'' \mid [\mathsf{0}]_{\{x\}})
\end{aligned}$$

As $(\nu\vec{a}\vec{y}x)(\mathsf{M}'' \mid [\mathsf{0}]_{\{x\}})$ and $C[\mathsf{N}]$ are insensitive, we use reduction-closure and soundness to deduce $\mathcal{T} \vdash \mathsf{0} = (\nu z)[\mathsf{P}]_A$ for all P and A. Applying (1) gives a contradiction, so actually $\mathcal{T} \not\vdash \mathsf{M} = \mathsf{N}$.

(3) and (4) are immediate consequences of (2). Finally, for (5), clearly $\overline{x} \#_3 \overline{y}$ whenever $x \neq y$, but $[\overline{x}]_{\{x,y\}}$ and $[\overline{y}]_{\{x,y\}}$ are both insensitive and hence equated by all sound theories. Since sound theories exist (take id as a readily verifiable example), we have proved (5). □

DEFINITION 64 Let S be set of networks. An i-theory $\mathcal{T}$ $(i = 1, 2)$ *isolates* S if $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ and $\mathsf{M} \in S$ together imply $\mathsf{N} \in S$.

If $S = (S_n, S_t)$ is a pair of sets of networks (S_n) and processes (S_p). A 3-theory $\mathcal{T}$ *isolates* S if $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \in S_p$ together imply $\mathsf{Q} \in S_p$, and $\mathcal{T} \vdash \mathsf{M} = \mathsf{N}$ and $\mathsf{M} \in S_n$ together imply $\mathsf{N} \in S_n$.

LEMMA 44 *Let $\mathcal{T}$ be a sound i-theory $(i = 1, 2, 3)$.*

1. *Assume $i \neq 3$. If $\mathcal{T}$ isolates a S where S is neither empty or universal, then $\mathcal{T}$ is consistent.*

2. *Assume $i = 3$. If $\mathcal{T}$ isolates (S_n, S_t) where neither S_n nor S_t are empty or universal, then $\mathcal{T}$ is consistent.*

3. *If $\mathcal{T}_i$ isolates S for each $i \in I$ then $\Sigma_{i \in I} \mathcal{T}_i$ also isolates S.*

4. *Assume $i \neq 3$. Then $\mathcal{T}$ isolates $(\{\mathsf{M} | \mathsf{M} \Downarrow_x\}$ for each name x.*

5. *If $i = 3$, then $\mathcal{T}$ isolates $(\{\mathsf{M} | \mathsf{M} \Downarrow_x\}, \{\mathsf{P} | \mathsf{P} \Downarrow_x\})$.*

6. *Let $\mathcal{T}_j$ be a sound i-theory for each $j \in J$. Then $\Sigma_{j \in J} \mathcal{T}_j$ is also sound.*

PROOF: (1) is immediate from the definitions. (3) is immediate by Lemma 42.3, (4) is by Lemma 43.3. In the light of 42.4, we need only show that $\Sigma_{i \in I} \mathcal{T}_i$ is consistent to establish (6): choose $x \in \mathcal{N}$. By (4) we know that $\mathcal{T}_i$ isolates $\{\mathsf{P} \mid \mathsf{P} \Downarrow_x\}$, so $\Sigma_{i \in I} \mathcal{T}_i$ does the same by (3). Now (1) guarantees soundness. □

In §4.4 we defined the maximum sound theory with respect to the inclusion ordering on the induced sets of consequences. This immediately generalises to 1- and 2-theories, but 3-theories admit two immediately appealing orderings.

$$(\mathcal{T}_1, \mathcal{T}_1') \sqsubseteq_1 (\mathcal{T}_2, \mathcal{T}_2') \quad \text{iff} \quad |\mathcal{T}_1| \subseteq |\mathcal{T}_2| \text{ and } |\mathcal{T}_2| \subseteq |\mathcal{T}_2'|$$
$$(\mathcal{T}_1, \mathcal{T}_1') \sqsubseteq_2 (\mathcal{T}_2, \mathcal{T}_2') \quad \text{iff} \quad |\mathcal{T}_1| \subseteq |\mathcal{T}_2| \text{ and } |\mathcal{T}_1| = |\mathcal{T}_2| \implies |\mathcal{T}_2| \subseteq |\mathcal{T}_2'|$$

Fortunately, both definitions coincide at the top.

LEMMA 45 *If 3-theories $\mathcal{T}_i$ are maximum with respect to $\sqsubseteq_i$ $(i = 1, 2)$, then $\mathcal{T}_1 = \mathcal{T}_2$.*

PROOF: Immediate from the definitions. □

DEFINITION 65 For 3-theories $\mathcal{T}, \mathcal{T}'$ we write $\mathcal{T} \sqsubseteq \mathcal{T}'$ iff $\mathcal{T} \sqsubseteq_1 \mathcal{T}'$. For 1-theories and 2-theories, the order $\sqsubseteq$ is simply $\subseteq$.

We can now complete the proof of Theorem 27

PROOF: For $i = 1, 2, 3$, Theorem 27.i is established exactly as the corresponding statement in Theorem 18, using the closure of sound i-theories under Σ (Lemmas 42 and 44). For the absence of a maximum reduction-closed consistent i-theory, we start with $i = 1$. Let $\mathsf{I} = [(\nu x)\overline{x}\langle\vec{y}\rangle]_\emptyset$ and define the 1-theory

$$\mathcal{T} = \{(\mathsf{M}|\mathsf{I}, \mathsf{N}|\mathsf{I}) \mid \mathsf{M}, \mathsf{N} \text{ networks}\}.$$

A contradiction is derived from the assumption that a maximum consistent and reduction-closed 1-theory exists, just as in Lemma 18, using Example 9.

For $i = 2$, define

$$\mathcal{T}' = \{([\mathsf{P}]_A, [\mathsf{Q}]_A) \mid \mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}\}$$

Since $\mathcal{T}$ and $\mathcal{T}'$ preserve weak barbs, as is easy to verify, both relations are consistent (Lemma 43). By a straightforward induction on the derivation of $\mathcal{T} \cup \mathcal{T}' \vdash \mathsf{M} = \mathsf{N}$ we show that $\mathcal{T} \cup \mathcal{T}' \not\vdash \mathsf{0} = \overline{a}\langle\vec{b}\rangle$, implying consistency. It is also straightforward to check that $\mathcal{T}'$ is reduction-closed (cf. the proof of Theorem 27). By Lemma 42.4, this means that $\mathcal{T} \cup \mathcal{T}'$ is reduction-closed. But $\overset{rc}{\approx}$ is a sound 1-theory (see Example 11), so $\mathsf{M} \overset{rc}{\approx} \mathsf{M}|\mathsf{I} \overset{rc}{\approx} \mathsf{N}|\mathsf{I} \overset{rc}{\approx} \mathsf{N}$ for all M, N, in violation to the assumed consistency.

Finally, for $i = 3$, we note that every 2-theory $\mathcal{T}$ induces a 3-theory $(\mathcal{T}, \mathcal{T}_{max})$. The lack of a maximum reduction-closed and consistent 2-theory then means that there cannot be a maximum reduction-closed and consistent 3-theory either. □

5.5 Reduction-Based Characterisations of $\mathcal{T}_{max}$

$\mathcal{T}_{max}$ admits characterisations as barbed congruence and along the lines of §4.5.2 and §4.5.3, but we will only describe the first of these here.

THEOREM 28 $|\,\mathcal{T}_{\max}\,| = \mathcal{T}_{\max} = \overset{rc}{\approx}$ *and* $\overset{rc}{\sim} \subset \mathcal{T}_{\max}$.

PROOF: Obviously $|\,\mathcal{T}_{max}\,| = \mathcal{T}_{max}$. In Example 11 we have shown that $\overset{rc}{\approx}$ is a sound 1-theory. Conversely, clearly $\mathcal{T}_{max}$ is a barbed bisimulation (preservation of weak barbs is by Lemma 43.2). To see that $\mathcal{T}_{max} \neq \overset{rc}{\sim}$ consider the networks $[\overline{x}]_\emptyset$ and $[(\nu a)(\overline{a} \mid a.\overline{x})]_\emptyset$. They are equated by $\mathcal{T}_{max}$ (Example 10) but not by $\overset{rc}{\sim}$. □

Theorem 28 justifies our choice of barb.

5.6 A Labelled Approximisation to $\mathcal{T}_{max}$

The next step is to provide tractable tools for reasoning about the maximum sound theory. We will follow the established pattern and present a labelled transition system such that the induced notion of weak bisimilarity soundly approximates $\mathcal{T}_{max}$. Unfortunately, we could not come up with a labelled characterisation.

Asynchronous labelled transitions $\stackrel{l}{\rightarrow}_a$ are given in Figure 5.3. They are derived from the labelled transitions in Figures 5.2 by throwing out observations on processes. The next theorem verifies that $\stackrel{\tau}{\rightarrow}_a$ characterises π_{mlt}-computations.

THEOREM 29 *1.* $\mathsf{M} \rightarrow \mathsf{N}$ *iff* $\mathsf{M} \stackrel{\tau}{\rightarrow}_a \mathsf{N}$.

2. If $\mathsf{M} \stackrel{l}{\rightarrow} \mathsf{N}$ *then* $\mathsf{M} \underbrace{\stackrel{\tau}{\rightarrow}_a \cdots \stackrel{\tau}{\rightarrow}_a}_{\geq 0} \stackrel{l}{\rightarrow}_a \underbrace{\stackrel{\tau}{\rightarrow}_a \cdots \stackrel{\tau}{\rightarrow}_a}_{\geq 0} \mathsf{N}$.

PROOF: Straightforward. □

The following facts will be useful later.

LEMMA 46 *1. If* $\mathsf{M} \stackrel{l}{\rightarrow}_a \mathsf{N}$ *then* $\mathsf{ap}(\mathsf{M}) = \mathsf{ap}(\mathsf{N})$.

2. If $\mathsf{M} \stackrel{x(\vec{y})}{\longrightarrow}_a \mathsf{N}$ *then* $\mathsf{N} \equiv \mathsf{M} \,|\, \overline{x}\langle \vec{y} \rangle$.

3. If $\mathsf{M} \stackrel{\overline{x}\langle (\nu \vec{y}) \vec{z} \rangle}{\longrightarrow}_a \mathsf{N}$ *then* $\mathsf{M} \equiv (\nu \vec{y})(\mathsf{M}' \,|\, \overline{x}\langle \vec{z} \rangle)$, $\{\vec{y}\} \subseteq \{\vec{z}\} \setminus \{x\}$ *and* $\mathsf{N} \equiv \mathsf{M}'$.

PROOF: By a straightforward induction on the derivation of transitions. □

5.6.1 (Strong) Synchronous and Asynchronous Bisimulations

Next we define labelled bisimulations. One could now discuss the various ways this could be done, as we did in the section on reduction congruence, but to make things simple we just go with one.

DEFINITION 66 A binary symmetric relation $\mathcal{R}$ on networks is a *strong synchronous bisimulation* if $(\mathsf{M}, \mathsf{N}) \in \mathcal{R}$ and $\mathsf{M} \stackrel{l}{\rightarrow}_t \mathsf{M}'$ implies that some transition $\mathsf{N} \stackrel{l}{\rightarrow}_t \mathsf{N}'$ exists with $(\mathsf{M}', \mathsf{N}') \in \mathcal{R}$. Here t is either n or ϵ. The largest strong synchronous bisimulation is called *strong synchronous bisimilarity* and denoted $\sim$.

$\mathcal{R}$ is a *synchronous bisimulation* if $(\mathsf{M}, \mathsf{N}) \in \mathcal{R}$ and $\mathsf{M} \stackrel{\tau}{\rightarrow} \mathsf{M}'$ implies that some transition sequence $\mathsf{N} \underbrace{\stackrel{\tau}{\rightarrow} \cdots \stackrel{\tau}{\rightarrow}}_{\geq 0} \mathsf{N}'$ exists with $(\mathsf{M}', \mathsf{N}') \in \mathcal{R}$, and $\mathsf{M} \stackrel{l}{\rightarrow} \mathsf{M}'$ $(l \neq \tau)$ implies that some transition sequence $\mathsf{N} \underbrace{\stackrel{\tau}{\rightarrow} \cdots \stackrel{\tau}{\rightarrow}}_{\geq 0} \stackrel{l}{\rightarrow}_t \underbrace{\stackrel{\tau}{\rightarrow} \cdots \stackrel{\tau}{\rightarrow}}_{\geq 0} \mathsf{N}'$ exists with $(\mathsf{M}', \mathsf{N}') \in \mathcal{R}$ (t as above). The largest synchronous bisimulation is called *synchronous bisimilarity* and denoted $\approx$.

$$\text{(Intra)}\quad \frac{\mathsf{P} \xrightarrow{\tau}_a \mathsf{Q}}{[\mathsf{P}]_A \xrightarrow{\tau}_a [\mathsf{Q}]_A}$$

$$\text{(Send)}\quad \frac{x \notin A}{[\mathsf{P}|\overline{x}\langle\vec{y}\rangle]_A \xrightarrow{\tau}_a [\phi(\mathsf{P})]_A \mid \overline{x}\langle\vec{y}\rangle}$$

$$\text{(Get)}\quad \frac{x \in A}{[\mathsf{P}]_A \mid \overline{x}\langle\vec{y}\rangle \xrightarrow{\tau}_a [\mathsf{P}|\overline{x}\langle\vec{y}\rangle]_A}$$

$$\text{(Loss)}\quad \frac{}{\overline{x}\langle\vec{z}\rangle \xrightarrow{\tau}_a \mathsf{0}}$$

$$\text{(Dupl)}\quad \frac{}{\overline{x}\langle\vec{z}\rangle \xrightarrow{\tau}_a \overline{x}\langle\vec{z}\rangle \mid \overline{x}\langle\vec{z}\rangle}$$

$$\text{(EtherIn)}\quad \frac{}{\mathsf{0} \xrightarrow{x(\vec{y})}_a \overline{x}\langle\vec{y}\rangle}$$

$$\text{(EtherOut)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{0}}$$

$$\text{(N-Par)}\quad \frac{\mathsf{M} \xrightarrow{l}_a \mathsf{M}' \quad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{N}) = \emptyset \quad l = \overline{x}\langle(\nu\vec{y})\vec{z}\rangle \Rightarrow x \notin \mathsf{ap}(\mathsf{N})}{\mathsf{M}|\mathsf{N} \xrightarrow{l}_a \mathsf{M}'|\mathsf{N}}$$

$$\text{(N-Res)}\quad \frac{\mathsf{M} \xrightarrow{l}_a \mathsf{N} \quad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)}{(\nu x)\mathsf{M} \xrightarrow{l}_a (\nu x)\mathsf{N}}$$

$$\text{(N-Open)}\quad \frac{\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{N} \quad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle}_a \mathsf{N}}$$

$$\text{(N-Cong)}\quad \frac{\mathsf{M} \equiv \mathsf{M}' \quad \mathsf{M}' \xrightarrow{l}_a \mathsf{N}' \quad \mathsf{N} \equiv \mathsf{N}'}{\mathsf{M} \xrightarrow{l}_a \mathsf{N}}$$

Figure 5.3: An asynchronous transition system for the timed, asynchronous π-calculus with locations, message loss and message duplication. Transitions for processes are omitted and can be found in Figure 4.2.

By replacing $\stackrel{l}{\rightarrow}$ with $\stackrel{l}{\rightarrow}_a$ we obtain *(strong) asynchronous bisimulations*. Since asynchronous bisimulations are clearly closed under arbitrary unions and since $\equiv$ is an asynchronous bisimulation, the largest asynchronous bisimulation exists and we denote it by $\approx_a$.

LEMMA 47 *If $\mathcal{R}$ is an asynchronous bisimulation then $\mathsf{M}\,\mathcal{R}\,\mathsf{N}$ implies $\mathsf{ap}(\mathsf{M}) = \mathsf{ap}(\mathsf{N})$.*

PROOF: Assume $x \in \mathsf{ap}(\mathsf{N}) \setminus \mathsf{ap}(\mathsf{M})$. Then

$$\mathsf{M} \xrightarrow{x(\vec{y})}_a \mathsf{M} \,|\, \overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{M}$$

But $\mathsf{N} \stackrel{\tau}{\rightarrow}_a \cdots \stackrel{\tau}{\rightarrow}_a \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \stackrel{\tau}{\rightarrow}_a \cdots \stackrel{\tau}{\rightarrow}_a \mathsf{N}'$ implies $x \in \mathsf{ap}(\mathsf{N}')$ (Lemma 46). By a straightforward induction on the derivation of asynchronous transitions we show that this means that there cannot be a transition $\mathsf{N}' \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{N}''$. Hence M and N cannot be related by $\mathcal{R}$, contradicting the assumptions. □

The following theorem summarises some of $\approx_a$'s important properties.

THEOREM 30 *1. $\approx_a$ is a congruence*

2. $\approx_a$ is not closed under renaming.

3. $\sim \subset \approx \subset \approx_a \subset \mathcal{T}_{\max}$ and $\sim_a \subseteq \approx_a$.

4. If $\mathcal{T}_{\max} \vdash \mathsf{P} = \mathsf{Q}$ then $[\mathsf{P}]_A \sim_a [\mathsf{Q}]_A$.

PROOF: For (1), the only mildly non-trivial case is closure under parallel composition. So assume $\mathsf{M}\,\mathcal{R}\,\mathsf{N}$ and $\mathsf{M} \stackrel{l}{\rightarrow}_a \mathsf{M}'$ and proceed by induction on the derivation of this transition. If the last rule is: $\mathsf{M}|\mathsf{L} = [\mathsf{P}]_A \,|\, \overline{x}\langle\vec{y}\rangle \stackrel{\tau}{\rightarrow}_a [\mathsf{P} \,|\, \overline{x}\langle\vec{y}\rangle]_A$ because $x \in A$, then consider

$$[\mathsf{P}]_A \xrightarrow{x(\vec{y})}_a [\mathsf{P}]_A \,|\, \overline{x}\langle\vec{y}\rangle \stackrel{\tau}{\rightarrow}_a [\mathsf{P} \,|\, \overline{x}\langle\vec{y}\rangle]_A$$

By assumption we can find

$$\mathsf{N} \stackrel{\tau}{\rightarrow}_a \cdots \stackrel{\tau}{\rightarrow}_a \mathsf{N}' \xrightarrow{x(\vec{y})}_a \mathsf{N}'' \stackrel{\tau}{\rightarrow}_a \cdots \stackrel{\tau}{\rightarrow}_a \mathsf{N}'''$$

with $\mathsf{N}''' \approx_a [\mathsf{P} \,|\, \overline{x}\langle\vec{y}\rangle]_A$. By Lemma 46 then also

$$\mathsf{N} \stackrel{\tau}{\rightarrow}_a \cdots \stackrel{\tau}{\rightarrow}_a \mathsf{N}' \,|\, \overline{x}\langle\vec{y}\rangle \equiv \mathsf{N}'' \stackrel{\tau}{\rightarrow}_a \cdots \stackrel{\tau}{\rightarrow}_a \mathsf{N}''',$$

as required. The other cases are similar.

For (2) simply consider the processes P and Q in Lemma 34 and locate them appropriately: clearly $[\mathsf{P}]_{\{y\}} \approx_a [\mathsf{Q}]_{\{y\}}$ but $[\mathsf{P}]_{\{y\}}\{x/y\} \not\Downarrow_b$ while $[\mathsf{Q}]_{\{y\}}\{x/y\} \Downarrow_b$.

Theorem 29.2 implies that $\approx$ is an asynchronous bisimulation. To see that the inclusion is proper, note that $[\mathsf{fw}_{xx}]_{\{x\}} \not\approx [\mathsf{0}]_{\{x\}}$ because the synchronous transition

$[\mathsf{fw}_{xx}]_{\{x\}} \xrightarrow{x(\vec{y})} [\mathsf{fw}_{xx} \,|\, \overline{x}\langle\vec{y}\rangle]_{\{x\}}$ cannot be matched by $[\mathsf{0}]_{\{x\}}$. Next, define $\mathcal{R}$ up to $\equiv$ by

$$[\mathsf{fw}_{xx} \,|\, \Pi_{i\in I}\overline{x}\langle\vec{y_i}\rangle]_{\{x\}} \,|\, \Pi_{j\in J}\overline{a_j}\langle\vec{y_j}\rangle \;\; \mathcal{R} \;\; [\Pi_{i\in I}\overline{x}\langle\vec{y_i}\rangle]_{\{x\}} \,|\, \Pi_{j\in J}\overline{a_j}\langle\vec{y_j}\rangle.$$

where $I, J \subseteq \mathbb{N}$ are finite sets. It is straightforward to verify that $\mathcal{R}$ is an asynchronous bisimulation. Hence $[\mathsf{fw}_{xx}]_{\{x\}} \approx_a [\mathsf{0}]_{\{x\}}$. It is easy to verify that $\approx_a$ is a sound theory, that $\sim$ is a proper subset of $\approx$ and that $\sim_a$ is a proper subset of $\approx_a$.

To finish the proof of (3) we must show that $\mathcal{T}_{max}$ properly includes $\approx_a$. For that purpose consider

$$[\mathsf{P} \,|\, \mathsf{fw}_{ya} \,|\, \overline{x}\langle y\rangle]_A \qquad \text{and} \qquad [\mathsf{P} \,|\, \mathsf{fw}_{za} \,|\, \overline{x}\langle z\rangle]_A$$

where A contains y, z but not x and P is an arbitrary process such that $y, z \notin \mathsf{fn}(\mathsf{P})$. To see that these two networks are related by $\mathcal{T}_{max}$, define $\mathcal{T}$ by

$$[\mathsf{P}\{y/v\} \,|\, \mathsf{fw}_{yz} \,|\, \overline{x}\langle y\rangle]_A \,|\, \Pi_{i\in I}(\overline{b_i}\langle\vec{d_i}\rangle\{y/v\}) \;\; \mathcal{T} \;\; [\mathsf{P}\{z/v\} \,|\, \mathsf{fw}_{yz} \,|\, \overline{x}\langle y\rangle]_A \,|\, \Pi_{i\in I}(\overline{b_i}\langle\vec{d_i}\rangle\{z/v\})$$

$$[\mathsf{P}\{y/v\} \,|\, \mathsf{fw}_{yz}]_A \,|\, \Pi_{i\in I}(\overline{b_i}\langle\vec{d_i}\rangle\{y/v\}) \;\; \mathcal{T} \;\; [\mathsf{P}\{z/v\} \,|\, \mathsf{fw}_{yz}]_A \,|\, \Pi_{i\in I}(\overline{b_i}\langle\vec{d_i}\rangle\{z/v\})$$

A tedious but straightforward verification establishes that $\mathcal{T} \cup \{(\mathsf{M}, \mathsf{N}) \,|\, \mathsf{M}, \mathsf{N} \text{ insensitive}\}$ is a sound theory.

Finally, for (4) we define $\mathcal{R}$ up to $\equiv$ by

$$(\nu\vec{a})([\mathsf{P}]_A \,|\, \Pi_i\overline{a_i}\langle\vec{b_i}\rangle) \quad \mathcal{R} \quad (\nu\vec{a})([\mathsf{Q}]_A \,|\, \Pi_i\overline{a_i}\langle\vec{b_i}\rangle)$$

whenever $\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{Q}$. We omit the straightforward verification that $\mathcal{R}$ is a strong asynchronous bisimulation. □

5.7 Expansion Laws and Distribution of Processes

The remainder of this chapter is concerned with what may be one of the most important questions in the study of distributed systems: how to go from centralised to distributed computation? The form of the questions suggests that it is possible to apply the time-honoured engineering imperative of Divide-and-Conquer to the design of distributed algorithms. In what follows, we wish to convince the reader that this is indeed often possible, by distinguishing between an algorithmic core and its distribution scaffold. But what does that mean?

- The *algorithmic core* is what a program would look like if it didn't have to worry about the partial failures characteristic of distributed systems.
- The *distribution scaffold* describes all mechanism in place to mask those partial failures sufficiently, so the algorithmic core can work as planned. Examples of such mechanisms include the error correction that seek to eradicate byzantine faults, but in the present context the tricks for coping with message-loss and message-failure are more relevant.

In practise, it might not always be easy to determine the boundary between the two, but in the idealising world of process calculi, the initial question can be neatly summarised as: can we find a scaffolding transformation $[\![\cdot]\!]$ that allows to go from $\mathsf{P} \mid \mathsf{Q}$ to $[\,[\![\mathsf{P}]\!]\,]_A \mid [\,[\![\mathsf{Q}]\!]\,]_B$ in a semantics preserving way?

Unfortunately, it is likely that the answer must be a resounding "no" in general. For a start, an unreliable transport medium cannot be turned into a divergence free and reliable one with a bit of software magic. Does that mean there cannot be interesting subclasses of processes where such a transformation is nevertheless possible? The present author does not think so and part of the point of this thesis is to exhibit an interesting subclass to inspire further work in this direction.

To see how one would go about finding such a class, consider the issues that render the transition from $\mathsf{P} \mid \mathsf{Q}$ to $[\,[\![\mathsf{P}]\!]\,]_A \mid [\,[\![\mathsf{Q}]\!]\,]_B$ problematic. There is only one really: message failure, if P and Q interact. Without this interaction, nothing would go wrong. But restricting distribution to processes that do not interact is hardly an interesting choice. The next clue comes from the mechanism we have to deal with non-byzantine message failures. There are essentially three.

- Message loss is masked by resending of disappeared message.
- Message loss is detected with explicit acknowledgements and timers that trigger suitable recovery mechanisms if acknowledgements fail to arrive in time.
- Message duplication is defused through messages carrying unique identifiers and/or input-linearity or input-affinity. A process is *input-linear on* x if it can receive on x exactly once. If it can receive at most once, it is *input-affine on* x.

Unfortunately, we cannot explore much of the theory of input linearity/affinity here, but one crucial thing to notice is that processes that are input input-linear or input-affine on a channel x is that they are not affected by message duplication on x, only by message loss. But that means message loss can be masked by resending messages sufficiently often. Linearity/affinity ensures that we can simply discard superfluous messages, should there be any. If $!\overline{x}\langle\vec{y}\rangle$ abbreviates $(\nu a)(\overline{a} \mid !a.(\overline{x}\langle\vec{y}\rangle \mid \overline{a}))$, the process

$$(\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle) \mid (x(\vec{v}).\mathsf{Q} \mid \mathsf{R})$$

could be transformed into

$$[[\![\mathsf{P}]\!] \mid !\overline{x}\langle\vec{y}\rangle]_A \mid [x(\vec{v}).[\![\mathsf{Q}]\!] \mid [\![\mathsf{R}]\!]]_B$$

without changing the semantics, provided $[\![\cdot]\!]$ does the right thing on P, Q and R. We will not investigate this transformation further because it does not allow for much refinement. Instead, we will focus on *Recursive Timers*. They are networks

of the form

$$[\mathsf{P} \mid (\nu ab)(\overline{x}\langle \vec{y}a\rangle \mid \mathsf{timer}^{t'}(a,\overline{b}) \mid !b.(\overline{x}\langle \vec{y}a\rangle \mid \mathsf{timer}^{t}(a,\overline{b})))]_A$$
$$\mid$$
$$[x(\vec{v}a).\mathsf{Q} \mid \mathsf{R}]_B,$$

subject to some further constraints, like $x \notin \mathsf{fn}(\mathsf{P})$, P timer-free etc., that we state later. The idea behind Recursive Timers is to replace outputs $\overline{x}\langle \vec{y}\rangle$ in the algorithmic core by $\overline{x}\langle \vec{y}a\rangle$ where a is a fresh acknowledgement channel. Dually, input-linear or input-affine inputs $x(\vec{v}).\mathsf{P}$ are converted to $x(\vec{v}a).(\overline{a} \mid \mathsf{Q})$ to include the receiver sending an acknowledgement. The semantic effect of this transformation is the same as that of output replication above, but it allows for an important generalisation: the *n-timed-resends*. They are different from Recursive Timers in that they resend lost messages only n times, rather than potentially infinitely often. If an acknowledgement has not been received after n resends, an alternative process is launched, which – in many cases – would involve some sort of notification for the process that requested the remote communication. Moving n to its limits yields recursive timers and what may be called *naive distribution* as special cases. The latter simply takes $\mathsf{P} \mid \mathsf{Q}$ into $[\mathsf{P}]_A \mid [\mathsf{Q}]_B$, without any concern for message failures. The hope is that the various forms of the n-timed-resends share many behavioural properties in forms that would allow an easy transfer of results obtained for one form to others. The rest of this chapter mostly studies Recursive Timers because it is the simplest useful form of n-timed-resends. It is also indispensable for the 2PCP. An in-depth study of n-timed-resends is on its way but its findings will have to be reported elsewhere.

Towards Expansion Laws

This section gives sufficient conditions for legitimately replacing certain remote communications by non-determinism (Theorem 31 below). Because it allows to remove timers, it can be considered an Expansion Theorem. It will be vital for reasoning about Recursive Timers. Its proof is typical for what is to follow, in that it is essentially a brute force exploration of the all possible transitions. Alas we could not come up with smarter proofs. Brute-force proofs of such facts can probably not be improved on. We will start with proving some basic facts about $\approx_a$ that are heavily used later.

LEMMA 48 *1. Let* Q *be timer-free and* $x \notin \mathsf{fn}(\mathsf{Q})$. *Then*

(a) $[\overline{x}\langle \vec{y}\rangle \mid x(\vec{y}).\mathsf{P} \mid \mathsf{Q}]_A \approx_a [\mathsf{P}\{\vec{y}/\vec{v}\} \mid \mathsf{Q}]_A$.

(b) $[\overline{x}\langle \vec{y}\rangle \mid !x(\vec{y}).\mathsf{P} \mid \mathsf{Q}]_A \approx_a [\mathsf{P}\{\vec{y}/\vec{v}\} \mid !x(\vec{y}).\mathsf{P} \mid \mathsf{Q}]_A$.

2. If $x \notin \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{M})$ *then*

(a) $(\nu x)(\mathsf{M} \mid [\mathsf{P}]_A) \approx_a \mathsf{M} \mid [\mathsf{P}]_{A\setminus\{x\}}$.

(b) $(\nu x)(\mathsf{M} \mid [\mathsf{P} \mid !x(\vec{v}).\mathsf{Q}]_A) \approx_a (\nu x)(\mathsf{M} \mid [\mathsf{P}]_A)$.

(c) $(\nu x)(\mathsf{M} \mid [\mathsf{P} \mid x(\vec{v}).\mathsf{Q}]_A) \approx_a (\nu x)(\mathsf{M} \mid [\mathsf{P}]_A)$.

(d) *In addition, if* $\mathsf{R} \approx_a \mathsf{0}$, *then* $(\nu x)(\mathsf{M} \mid [\mathsf{P} \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R})]_A) \approx_a (\nu x)(\mathsf{M} \mid [\mathsf{P}]_A)$.

(e) $(\nu x)(\mathsf{M} \mid [\overline{x}\langle\vec{y}\rangle \mid \mathsf{P}]_A) \approx_a (\nu x)(\mathsf{M} \mid [\mathsf{P}]_A)$.

3. *Let* $n \geq 0$, *then (on the network level)* $(\nu x)\Pi_{i=1}^n \overline{x}\langle\vec{y}\rangle \approx_a \mathsf{0}$.

4. $\mathsf{M} \mid \Pi_{i=1}^m \overline{x}\langle\vec{y}\rangle \approx_a \mathsf{M} \mid \Pi_{i=1}^n \overline{x}\langle\vec{y}\rangle$ *for all* $m, n \geq 0$, *provided:* $m = 0 \Leftrightarrow n = 0$.

5. *If* $x \in A \setminus \mathsf{fn}(\mathsf{P})$, *then*

$$[\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle]_A \approx_a [\mathsf{P}]_A \mid \overline{x}\langle\vec{y}\rangle \approx_a [\mathsf{P}]_A.$$

6. $\Pi_{i=1}^n [\overline{x_i}\langle\vec{y_i}\rangle]_{A_i} \approx_a [\Pi_{i=1}^n \overline{x_i}\langle\vec{y_i}\rangle]_A$ *where* $A = \bigcup_{i=1}^n A_i$.

7. *If* $x \in B \setminus \mathsf{fin}(\mathsf{P})$, $C[\cdot]$ *is an arbitrary process context and* x *is not bound in* $C[\cdot]$, *then*

$$[\, C[\overline{x}\langle\vec{y}\rangle]\,]_A \mid [\mathsf{P}]_B \approx_a [\, C[\mathsf{0}]\,]_A \mid [\mathsf{P}]_B.$$

Proof: The proof for (1) is straightforward. Consider

$$[\overline{x}\langle\vec{y}\rangle \mid x(\vec{v}).\mathsf{P} \mid \mathsf{Q}]_A \mid \Pi_{i=1}^n \overline{a_i}\langle\vec{b_i}\rangle \;\; \mathcal{R} \;\; [\mathsf{P}\{\vec{y}/\vec{v}\} \mid \mathsf{Q}]_A \mid \Pi_{i=1}^n \overline{a_i}\langle\vec{b_i}\rangle.$$

where $n \geq 0$, a_i is a name, $\vec{b_i}$ a vector of names and Q is a timer-free process not containing x. Clearly $\mathcal{R} \cup \mathsf{id}$ is an asynchronous bisimulation, proving (1a). The other case is similar. (2, 3, 4) are also straightforward and omitted. The key insight for proving (5) is that $x \in A$ prevents asynchronous transitions $\xrightarrow{\overline{x}\langle\vec{y}\rangle}_a$. A witness $\mathcal{R}$ for (6) is given by

$$\Pi_{i=1}^n [\overline{x_i}\langle\vec{y_i}\rangle \mid \Pi_{j=1, a_j \in A_i}^m \overline{a_j}\langle\vec{b_j}\rangle]_{A_i} \mid \Pi_{j=1}^k \overline{c_j}\langle\vec{d_j}\rangle$$
$$\mathcal{R}$$
$$[\Pi_{i=1}^n \overline{x_i}\langle\vec{y_i}\rangle \mid \Pi_{j=1}^m \overline{a_j}\langle\vec{b_j}\rangle]_A \mid \Pi_{j=1}^k \overline{c_j}\langle\vec{d_j}\rangle$$

where $k, m \geq 0$. It is easy to verify that $\mathcal{R}$ is indeed an asynchronous bisimulation, taking into account that $\{A_i\}_{i=1}^n$ partitions A.

The proof of (7) proceeds by induction on the structure of $C[\cdot]$. We only deal with the most complicated case $C[\cdot] = a(\vec{v}).C'[\cdot]$ where $x \notin \{\vec{v}\}$. The witness relation $\mathcal{R}$ is given as

$$(\nu\vec{e})([\Pi_{i=i}^m \overline{x_i}\langle\vec{y_i}\rangle \mid a(\vec{v}).C'[\overline{x}\langle\vec{y}\rangle]]_A \mid \Pi_{i=i}^n \overline{a_i}\langle\vec{b_i}\rangle \mid [\Pi_{i=i}^k \overline{c_i}\langle\vec{d_i}\rangle \mid \mathsf{P}]_B)$$
$$\mathcal{R}$$
$$(\nu\vec{e})([\Pi_{i=i}^m \overline{x_i}\langle\vec{y_i}\rangle \mid a(\vec{v}).C'[\mathsf{0}]]_A \mid \Pi_{i=i}^n \overline{a_i}\langle\vec{b_i}\rangle \mid [\Pi_{i=i}^k \overline{c_i}\langle\vec{d_i}\rangle \mid \mathsf{P}]_B)$$

We only discuss one transition that could arise from $\mathcal{R} \cup \approx_a$. The remaining ones are easier. If $m \geq 1$ and w.l.o.g. $x_m = a$, then we have a transition (omitting the outermost restriction)

$$\begin{aligned}&[\Pi_{i=i}^{m} \overline{x_i}\langle \vec{y_i} \rangle \mid a(\vec{v}).C'[\overline{x}\langle \vec{y} \rangle]]_A \mid \Pi_{i=i}^{n} \overline{a_i}\langle \vec{b_i} \rangle \mid [\Pi_{i=i}^{k} \overline{c_i}\langle \vec{d_i} \rangle \mid \mathsf{P}]_B \\ &\quad \xrightarrow{\tau}_a [\Pi_{i=i}^{m-1} \overline{x_i}\langle \vec{y_i} \rangle \mid C'[\overline{x}\langle \vec{y} \rangle]\{\vec{b_m}/\vec{v}\}]_A \mid \Pi_{i=i}^{n} \overline{a_i}\langle \vec{b_i} \rangle \mid [\Pi_{i=i}^{k} \overline{c_i}\langle \vec{d_i} \rangle \mid \mathsf{P}]_B\end{aligned}$$

It is matched by

$$\begin{aligned}&[\Pi_{i=i}^{m} \overline{x_i}\langle \vec{y_i} \rangle \mid a(\vec{v}).C'[\mathsf{0}]]_A \mid \Pi_{i=i}^{n} \overline{a_i}\langle \vec{b_i} \rangle \mid [\Pi_{i=i}^{k} \overline{c_i}\langle \vec{d_i} \rangle \mid \mathsf{P}]_B \\ &\quad \xrightarrow{\tau}_a [\Pi_{i=i}^{m-1} \overline{x_i}\langle \vec{y_i} \rangle \mid C'[\mathsf{0}]\{\vec{b_m}/\vec{v}\}]_A \mid \Pi_{i=i}^{n} \overline{a_i}\langle \vec{b_i} \rangle \mid [\Pi_{i=i}^{k} \overline{c_i}\langle \vec{d_i} \rangle \mid \mathsf{P}]_B\end{aligned}$$

by (IH). □

The next lemma shows that networks, input-linear on x, need not worry about message duplication.

LEMMA 49 *Let $x \in A$ but $x \notin \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q}) \cup \mathsf{fn}(\mathsf{R})$. Also assume that $x(\vec{v}).\mathsf{Q}$ is local. Then for all $m, n \geq 0$ such that $m = 0 \Leftrightarrow n = 0$:*

1. $[\mathsf{P} \mid x(\vec{v}).\mathsf{Q} \mid \Pi_{i=1}^{m} \overline{x}\langle \vec{y_i} \rangle]_A \approx_a [\mathsf{P} \mid x(\vec{v}).\mathsf{Q} \mid \Pi_{i=1}^{n} \overline{x}\langle \vec{y_i} \rangle]_A$.
2. $[\mathsf{P} \mid !x(\vec{v}).\mathsf{Q} \mid \Pi_{i=1}^{m} \overline{x}\langle \vec{y_i} \rangle]_A \approx_a [\mathsf{P} \mid x(\vec{v}).\mathsf{Q} \mid \Pi_{i=1}^{n} \overline{x}\langle \vec{y_i} \rangle]_A$.
3. $[\mathsf{P} \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R}) \mid \Pi_{i=1}^{m} \overline{x}\langle \vec{y_i} \rangle]_A \approx_a [\mathsf{P} \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R}) \mid \Pi_{i=1}^{n} \overline{x}\langle \vec{y_i} \rangle]_A$.

PROOF: For (1), we define $\mathcal{R}$ by

$$[\mathsf{P} \mid x(\vec{v}).\mathsf{Q} \mid \Pi_{i=1}^{m} \overline{x}\langle \vec{y_i} \rangle]_A \mid \Pi_{i=1}^{k} \overline{a_i}\langle \vec{b_i} \rangle \;\; \mathcal{R} \;\; [\mathsf{P} \mid x(\vec{v}).\mathsf{Q} \mid \Pi_{i=1}^{n} \overline{x}\langle \vec{y_i} \rangle]_A \mid \Pi_{i=1}^{k} \overline{a_i}\langle \vec{b_i} \rangle$$

and show that $\mathcal{R} \cup \approx_a$ is an asynchronous bisimulation. This is straightforward. The other two cases are essentially similar. □

we can now prove the first expansion theorem for π_{mlt}. As pointed out before, it allows to reduce timers to internal non-determinism, provided the channel that can be used to stop the timer is input-linear.

THEOREM 31 *(Expansion)*

Assume P and S are timer-free and x is fresh. Then for all $k, m, n \geq 0$ with $k + m + n > 0$:

$$\begin{gathered}(\nu x)(\mathsf{M} \mid [\mathsf{P} \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R}) \mid \Pi_{i=1}^{k} \overline{x}\langle \vec{y} \rangle]_{A \cup \{x\}} \mid \Pi_{i=1}^{m} \overline{x}\langle \vec{y} \rangle \mid \; [\Pi_{i=1}^{n} \overline{x}\langle \vec{y} \rangle \mid \mathsf{S}]_B) \\ \approx_a \\ \mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B.\end{gathered}$$

PROOF: We use the following abbreviation.

$$\mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) = (\nu x)(\mathsf{M} \mid [\mathsf{P} \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{Q},\mathsf{R}) \mid \Pi^k_{i=1}\overline{x}\langle\vec{y}\rangle]_{A\cup\{x\}} \mid \Pi^m_{i=1}\overline{x}\langle\vec{y}\rangle \mid [\Pi^n_{i=1}\overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B)$$

Define $\mathcal{R}$ by

$$(\nu\vec{c})\mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) \quad \mathcal{R} \quad (\nu\vec{c})(\mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B)$$

where P and S range over all suitable processes, M ranges over all suitable networks and $0 < t$. Then $\mathcal{R} \cup \approx_a$ is an asynchronous bisimulation. To see this, we check all relevant transitions, ignoring $(\nu\vec{c})$ and bound outputs for simplicity. It is trivial to fill in the missing details.

- If $\mathsf{P} \xrightarrow{\tau}_a \mathsf{P}'$ and hence

  $$\mathsf{N}^{t+1}_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) \xrightarrow{\tau}_a \mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P}',\mathsf{S})$$

  then

  $$\mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \xrightarrow{\tau}_a \mathsf{M} \mid [\mathsf{P}' \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B$$

  is matching because timer-freeness and the absence of a given name is preserved by $\xrightarrow{\tau}_a$-transitions.

- If $\mathsf{P} \xrightarrow{\tau}_a \mathsf{P}'$ and hence

  $$\mathsf{N}^1_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) \xrightarrow{\tau}_a (\nu x)(\mathsf{M} \mid [\mathsf{P}' \mid \mathsf{R} \mid \Pi^k_{i=1}\overline{x}\langle\vec{y}\rangle]_{A\cup\{x\}} \mid \Pi^m_{i=1}\overline{x}\langle\vec{y}\rangle \mid [\Pi^n_{i=1}\overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B)$$

  then

  $$\mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid \Pi^n_{i=1}\overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \xrightarrow{\tau}_a \mathsf{M} \mid [\mathsf{P}' \mid \mathsf{R}]_A \mid [\mathsf{S}]_B$$

  matches because

  $$(\nu x)(\mathsf{M} \mid [\mathsf{P}' \mid \mathsf{R} \mid \Pi^k_{i=1}\overline{x}\langle\vec{y}\rangle]_{A\cup\{x\}} \mid \Pi^m_{i=1}\overline{x}\langle\vec{y}\rangle \mid [\Pi^n_{i=1}\overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B) \quad \approx_a \quad \mathsf{M} \mid [\mathsf{P}' \mid \mathsf{R}]_A \mid [\mathsf{S}]_B$$

  by Lemma 48.2 and 48.5.

- Transitions induced by $\mathsf{S} \xrightarrow{\tau}_a \mathsf{S}'$ or $\mathsf{M} \xrightarrow{l}_a \mathsf{M}'$ are dealt with as in the previous two cases.

- If $a \notin A \cup \{x\}$ and

  $$\mathsf{N}^{t+1}_{kmn}(\mathsf{M},\mathsf{P}|\overline{a}\langle\vec{b}\rangle,\mathsf{S}) \xrightarrow{\tau}_a \mathsf{N}^t_{kmn}(\mathsf{M}|\overline{a}\langle\vec{b}\rangle,\mathsf{P},\mathsf{S})$$

  then

  $$\mathsf{M} \mid [\mathsf{P} \mid \overline{a}\langle\vec{b}\rangle \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \xrightarrow{\tau}_a \mathsf{M} \mid \overline{a}\langle\vec{b}\rangle \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B$$

  is a matching transition.

- If $a \notin A \cup \{x\}$ and

  $$\mathsf{N}^1_{kmn}(\mathsf{M}, \mathsf{P}|\overline{a}\langle\vec{b}\rangle, \mathsf{S}) \xrightarrow{\tau}_a (\nu x)(\mathsf{M} \mid \overline{a}\langle\vec{b}\rangle \mid [\mathsf{P} \mid \mathsf{R} \mid \Pi^k_{i=1}\overline{x}\langle\vec{y}\rangle]_{A\cup\{x\}} \mid \Pi^m_{i=1}\overline{x}\langle\vec{y}\rangle \mid [\Pi^n_{i=1}\overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B)$$

  then

  $$\mathsf{M} \mid [\mathsf{P} \mid \overline{a}\langle\vec{b}\rangle \mid (\mathsf{P}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid \Pi^n_{i=1}\overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \xrightarrow{\tau}_a \mathsf{M} \mid \overline{a}\langle\vec{b}\rangle \mid [\mathsf{P} \mid \mathsf{R}]_A \mid [\mathsf{S}]_B$$

  is a match, using Lemma 48.2 and 48.5 again.

- If $a \in A$, $a \neq x$ and

  $$\mathsf{N}^t_{kmn}(\mathsf{M}|\overline{a}\langle\vec{b}\rangle, \mathsf{P}, \mathsf{S}) \xrightarrow{\tau}_a \mathsf{N}^t_{kmn}(\mathsf{M}, \mathsf{P}|\overline{a}\langle\vec{b}\rangle, \mathsf{S})$$

  then

  $$\mathsf{M} \mid \overline{a}\langle\vec{b}\rangle \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B \xrightarrow{\tau}_a \mathsf{M} \mid [\mathsf{P} \mid \overline{a}\langle\vec{b}\rangle \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B$$

  is a match.

- If $\mathsf{N}^t_{kmn}(\mathsf{M}, \mathsf{P}, \mathsf{S}) \xrightarrow{l}_a \mathsf{N}^t_{kmn}(\mathsf{M}', \mathsf{P}, \mathsf{S}')$ because $\mathsf{M} \mid [\mathsf{S}]_B \xrightarrow{l}_a \mathsf{M}' \mid [\mathsf{S}']_B$ (please remember that we omit scope extension), then

  $$\mathsf{M} \mid \overline{a}\langle\vec{b}\rangle \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B \xrightarrow{\tau}_a \mathsf{M}' \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}']_B$$

  is a match.

- If we have a message duplication

  $$\mathsf{N}^t_{km+1n}(\mathsf{M}, \mathsf{P}, \mathsf{S}) \xrightarrow{\tau}_a \mathsf{N}^t_{km+2n}(\mathsf{M}', \mathsf{P}, \mathsf{S}')$$

  then the empty transition sequence is a match.

- For a message loss

  $$\mathsf{N}^t_{km+1n}(\mathsf{M}, \mathsf{P}, \mathsf{S}) \xrightarrow{l}_a \mathsf{N}^t_{kmn}(\mathsf{M}', \mathsf{P}, \mathsf{S}')$$

  we have two cases. If $k + m + n = 0$, then

  $$\mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B \xrightarrow{\tau}_a \mathsf{M} \mid [\mathsf{P} \mid \mathsf{R}]_A \mid [\mathsf{S}]_B$$

  is a match, using Lemma 48.2 and 48.5, otherwise the empty transition sequence does the job.

- Message migration

  $$\mathsf{N}^t_{kmn+1}(\mathsf{M}, \mathsf{P}, \mathsf{S}) \xrightarrow{\tau}_a \mathsf{N}^t_{km+1n}(\mathsf{M}', \mathsf{P}, \mathsf{S}')$$

  or

  $$\mathsf{N}^t_{km+1n}(\mathsf{M}, \mathsf{P}, \mathsf{S}) \xrightarrow{\tau}_a \mathsf{N}^t_{k+1mn}(\mathsf{M}', \mathsf{P}, \mathsf{S}')$$

  can also be matched by the empty transition sequence.

- A time-out interaction

  $$\mathsf{N}^t_{k+1mn}(\mathsf{M},\mathsf{P},\mathsf{S}) \xrightarrow{\tau}_a (\nu x)(\mathsf{M} \mid [\mathsf{P} \mid \mathsf{Q}\{\vec{y}/\vec{v}\} \mid \Pi^k_{i=1}\overline{x}\langle\vec{y}\rangle]_{A\cup\{x\}} \mid \Pi^m_{i=1}\overline{x}\langle\vec{y}\rangle \mid [\Pi^n_{i=1}\overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B)$$

  is matched by

  $$\mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B \xrightarrow{\tau}_a \mathsf{M} \mid [\mathsf{P} \mid \mathsf{Q}\{\vec{y}/\vec{v}\}]_A \mid [\mathsf{S}]_B$$

  using Lemma 48.2 and 48.5 again.

- An input action

  $$\mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) \xrightarrow{a(\vec{b})}_a \mathsf{N}^t_{kmn}(\mathsf{M}|\overline{a}\langle\vec{b}\rangle,\mathsf{P},\mathsf{S})$$

  where $a \neq x$ can be matched by

  $$\mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B \xrightarrow{a(\vec{b})}_a \mathsf{M} \mid \overline{a}\langle\vec{b}\rangle \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B.$$

- If $a \neq x$ then the output

  $$\mathsf{N}^t_{kmn}(\mathsf{M}|\overline{a}\langle\vec{b}\rangle,\mathsf{P},\mathsf{S}) \xrightarrow{\overline{a}\langle\vec{b}\rangle}_a \mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S})$$

  is matched by

  $$\mathsf{M} \mid \overline{a}\langle\vec{b}\rangle \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B \xrightarrow{\overline{a}\langle\vec{b}\rangle}_a \mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B.$$

Now we must consider the transitions from the right component of $\mathcal{R}$.

- A match for the internal choice

  $$\mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B \xrightarrow{\tau}_a \mathsf{M} \mid [\mathsf{P} \mid \mathsf{Q}\{\vec{y}/\vec{v}\}]_A \mid [\mathsf{S}]_B$$

  has 3 cases. If $k > 0$, then

  $$\mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) \xrightarrow{\tau}_a (\nu x)(\mathsf{M} \mid [\mathsf{P} \mid \mathsf{Q}\{\vec{y}/\vec{v}\} \mid \Pi^{k-1}_{i=1}\overline{x}\langle\vec{y}\rangle]_{A\cup\{x\}} \mid \Pi^m_{i=1}\overline{x}\langle\vec{y}\rangle \mid [\Pi^n_{i=1}\overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B)$$

  is a match by Lemma 48.2 and 48.5. If $m > 0$, then

  $$\mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) \xrightarrow{\tau}_a \mathsf{N}^t_{k+1m-1n}(\mathsf{M},\mathsf{P},\mathsf{S})$$

  can be completed to a matching transition sequence with the transition given for $k = 0$. Finally, if $n > 0$, then

  $$\mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) \xrightarrow{\tau}_a \mathsf{N}^t_{km+1n-1}(\mathsf{M},\mathsf{P},\mathsf{S})$$

  can be completed as in the case $m > 0$.

- The transition

$$\mathsf{M} \mid [\mathsf{P} \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B \xrightarrow{\tau}_a \mathsf{M} \mid \mathsf{R} \mid [\mathsf{P} \mid \mathsf{R}]_A \mid [\mathsf{S}]_B$$

  is matched by

$$\mathsf{N}^t_{kmn}(\mathsf{M},\mathsf{P},\mathsf{S}) \underbrace{\xrightarrow{\tau}_a \cdots \xrightarrow{\tau}_a}_{t} (\nu x)(\mathsf{M} \mid [\mathsf{P} \mid \mathsf{R} \mid \Pi_{i=1}^k \overline{x}\langle\vec{y}\rangle]_{A\cup\{x\}} \mid \Pi_{i=1}^m \overline{x}\langle\vec{y}\rangle \mid [\Pi_{i=1}^n \overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B)$$

  with the help of Lemma 48.2 and 48.5.

The remaining transitions are straightforwardly matched in ways that have already been demonstrated by previous cases. □

Unfortunately, the expansion theorem is a bit too weak for our purposes. The problem is that the processes running in parallel with the timer that waits for an acknowledgement must be timer free. But in a Recursive Timer, a time-out triggers a new timer to be started. The following variant of the above Expansion Theorem is more liberal in this regard

THEOREM 32 *(Expansion (2)) Assume that the following is true.*

- P *and* S *are timer-free.*
- $x \notin \mathsf{fn}(\mathsf{M}) \cup \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{S}) \cup \{\vec{b}\vec{c}\}$.
- $\{\vec{b},\vec{c}\} \cap (\mathsf{fn}(\mathsf{M}) \cup \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{S})) = \emptyset$.

Then for all $m, m, n \geq 0$ *with* $k + m + n > 0$*:*

$$(\nu x\vec{b}\vec{c})(\mathsf{M} \mid [\mathsf{P} \mid \Pi_i b_i(\vec{v_i}).\mathsf{P}_i \mid \Pi_i !c_i(\vec{w_i}).\mathsf{P}'_i \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{Q},\mathsf{R}) \mid \Pi_{i=1}^k \overline{x}\langle\vec{y}\rangle]_A \mid \Pi_{i=1}^m \overline{x}\langle\vec{y}\rangle \mid\ [\Pi_{i=1}^n \overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B)$$

$$\approx_a$$

$$(\nu x\vec{b}\vec{c})(\mathsf{M} \mid [\mathsf{P} \mid \Pi_i b_i(\vec{v_i}).\mathsf{P}_i \mid \Pi_i !c_i(\vec{w_i}).\mathsf{P}'_i \mid (\mathsf{Q}\{\vec{y}/\vec{v}\} \oplus \mathsf{R})]_A \mid [\mathsf{S}]_B).$$

PROOF: The proof is essentially like that of Theorem 31. The main difference is the additional inputs $\Pi_i b_i(\vec{v_i}).\mathsf{P}_i \mid \Pi_i !c_i(\vec{w_i}).\mathsf{P}'_i$ that may contain timers. But the assumptions on $\vec{b}$ and $\vec{c}$ guarantee that only the processes Q and R may contain outputs that could activate timers in P_i and P'_i. Hence these additional inputs cannot add transitions that are of significant interest in the proof. □

5.7.1 Recursive Timers

With the expansion law out of the way, we can now investigate Recursive Timers. What we ought to show is the following. If $\vdash [\mathsf{P}]_A \mid [x(\vec{y}).\mathsf{Q}]_B$, $x \notin \mathsf{fn}(\mathsf{Q})$ and P as well as Q are timer free. Then

$$[\mathsf{P} \mid (\nu ab)(\overline{x}\langle\vec{y}a\rangle \mid \mathsf{timer}^t(a,\overline{b}) \mid !b.(\overline{x}\langle\vec{y}a\rangle \mid \mathsf{timer}^t(a,\overline{b}))]_A \mid [x(\vec{v}a).(\overline{a} \mid \mathsf{Q})]_B$$

$$\approx_a$$

$$[\mathsf{P}]_A \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B$$

(a, b are fresh). This equation can be summarised as "remote messages, guarded by Recursive Timers, will eventually get through". It should be understood that this holds only "up to divergence". It is perfectly possible for Recursive Timers to produce an infinite number of messages, none of which reaches its receiver. But $\approx_a$ and hence $\mathcal{T}_{max}$ is divergence insensitive, as one can easily establish. So the equality above really says "either the message get through eventually, or infinitely many messages get lost and none gets through". This may be seen as a rather weak result. But in realistic settings the probability that infinitely many messages get lost is often taken to be 0. This justifies the initial slogan "remote messages, guarded by Recursive Timers, will eventually get through". Having to rely on probabilistic arguments that are not reflected in the formal model is somewhat unsatisfactory, but probabilistic semantics for π-calculi are not well-understood, so for the time being we have to make do with conventional semantics and its hand-waving about how likely any given branches in a synchronisation tree may be.

While establishing the correctness of the above equation we use these two abbreviations:

$$\begin{aligned} \mathsf{timer}^0(x(\vec{v}).\mathsf{P}, \mathsf{Q}) &= \mathsf{Q}, \\ \mathsf{U}^s &= \mathsf{timer}^s(a, \overline{b}) \,|\, !b.(\overline{x}\langle \vec{y}a\rangle \,|\, \mathsf{timer}^t(a, \overline{b})) \end{aligned}$$

where $s \geq 0$. Then we would like to reason as follows.

$$\begin{aligned} & [\mathsf{P} \,|\, (\nu ab)(\overline{x}\langle \vec{y}a\rangle \,|\, \mathsf{U}^t)]_A \mid [x(v\vec{a}).(\overline{a} \,|\, \mathsf{Q})]_B \\ \equiv\ & (\nu ab)([\mathsf{P} \,|\, \overline{x}\langle \vec{y}a\rangle \,|\, \mathsf{U}^t]_{A\cup\{ab\}} \mid [x(v\vec{a}).(\overline{a} \,|\, \mathsf{Q})]_B) \\ \approx_a\ & (\nu ab)([\mathsf{P} \,|\, \mathsf{U}^t]_{A\cup\{ab\}} \,|\, \overline{x}\langle \vec{y}a\rangle \mid [x(\vec{v}a).(\overline{a} \,|\, \mathsf{Q})]_B) \\ \approx_a\ & (\nu ab)([\mathsf{P} \,|\, \mathsf{U}^t]_{A\cup\{ab\}} \mid [\overline{x}\langle \vec{y}a\rangle \,|\, x(\vec{v}a).(\overline{a} \,|\, \mathsf{Q})]_B) \\ \approx_a\ & (\nu ab)([\mathsf{P} \,|\, \mathsf{U}^t]_{A\cup\{ab\}} \mid [\overline{a} \,|\, \mathsf{Q}\{\vec{y}/\vec{v}\}]_B) \\ \approx_a\ & (\nu ab)([\mathsf{P} \,|\, \mathsf{U}^t]_{A\cup\{ab\}} \mid \overline{a} \,|\, [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) \\ \approx_a\ & (\nu ab)([\mathsf{P} \,|\, \mathsf{U}^t \,|\, \overline{a}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) \\ =\ & (\nu ab)([\mathsf{P} \,|\, !b.(\overline{x}\langle \vec{y}a\rangle \,|\, \mathsf{timer}^t(a, \overline{b})) \,|\, \mathsf{timer}^t(a, \overline{b}) \,|\, \overline{a}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) \\ \approx_a\ & (\nu ab)([\mathsf{P} \,|\, !b.(\overline{x}\langle \vec{y}a\rangle \,|\, \mathsf{timer}^t(a, \overline{b})) \,|\, \overline{b} \oplus \mathsf{0}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) \\ \approx_a\ & (\nu ab)([\mathsf{P} \,|\, !b.\mathsf{timer}^t(a, \overline{b}) \,|\, \overline{b} \oplus \mathsf{0}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) \\ \approx_a\ & (\nu ab)([\mathsf{P}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) \\ \equiv\ & [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B \,|\, (\nu ab)[\mathsf{P}]_{A\cup\{ab\}} \\ \approx_a\ & [\mathsf{P}]_A \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B \end{aligned}$$

Unfortunately, we could not establish several of the steps that would have made

this derivation valid. Instead we proceed with more coarse grained reasoning steps.

$$
\begin{aligned}
&[\mathsf{P} \mid (\nu ab)(\overline{x}\langle \vec{y}a\rangle \mid \mathsf{U}^t)]_A \mid [x(v\vec{a}).(\overline{a} \mid \mathsf{Q})]_B \\
&\quad \equiv \quad (\nu ab)([\mathsf{P} \mid \overline{x}\langle \vec{y}a\rangle \mid \mathsf{U}^t]_{A\cup\{ab\}} \mid [x(v\vec{a}).(\overline{a} \mid \mathsf{Q})]_B) \\
&\quad \approx_a \quad (\nu ab)([\mathsf{P} \mid \mathsf{U}^t]_{A\cup\{ab\}} \mid [\overline{x}\langle \vec{y}a\rangle \mid x(\vec{v}a).(\overline{a} \mid \mathsf{Q})]_B) && (5.4)\\
&\quad \approx_a \quad (\nu ab)([\mathsf{P} \mid \mathsf{U}^t]_{A\cup\{ab\}} \mid [\overline{a} \mid \mathsf{Q}\{\vec{y}/\vec{v}\}]_B) && (5.5)\\
&\quad \approx_a \quad (\nu ab)([\mathsf{P} \mid \mathsf{U}^t \mid \overline{a}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) && (5.6)\\
&\quad = \quad (\nu ab)([\mathsf{P} \mid !b.(\overline{x}\langle \vec{y}a\rangle \mid \mathsf{timer}^t(a,\overline{b})) \mid \mathsf{timer}^t(a,\overline{b}) \mid \overline{a}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) \\
&\quad \approx_a \quad (\nu ab)([\mathsf{P} \mid !b.(\overline{x}\langle \vec{y}a\rangle \mid \mathsf{timer}^t(a,\overline{b})) \mid \mathsf{0} \oplus \overline{b}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) && (5.7)\\
&\quad \approx_a \quad (\nu ab)([\mathsf{P} \mid !b.\mathsf{timer}^t(a,\overline{b}) \mid \mathsf{0} \oplus \overline{b}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) && (5.8)\\
&\quad \approx_a \quad (\nu ab)([\mathsf{P}]_{A\cup\{ab\}} \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B) && (5.9)\\
&\quad \equiv \quad [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B \mid (\nu ab)[\mathsf{P}]_{A\cup\{ab\}} \\
&\quad \approx_a \quad [\mathsf{P}]_A \mid [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B && (5.10)
\end{aligned}
$$

Here (5.5) is a consequence of Lemma 48.1, (5.10) follows directly from Lemma 48.2 and (5.7) is an application of the second Expansion Theorem (Theorem 32). This leave the remaining gaps (5.4), (5.6), (5.8) and (5.9) to be plugged. This is what we shall do now.

LEMMA 50 *Assume the following hold.*

- $m, m', m'', n, n', n'' \geq 0$,
- $0 \leq t', t'' \leq t$, $a \neq x$,
- P *timer-free and*
- $x \in B$.

With $\mathsf{Q}^s_k = \Pi_{i=1}^k \overline{x}\langle \vec{y}a\rangle \mid \mathsf{U}^s$ *we have*

$$[\mathsf{P} \mid \mathsf{Q}^{t'}_m]_A \mid \Pi_{i=1}^{m'} \overline{x}\langle \vec{y}\rangle \mid [\Pi_{i=1}^{m''} \overline{x}\langle \vec{y}\rangle \mid \mathsf{R}]_B \;\approx_a\; [\mathsf{P} \mid \mathsf{Q}^{t''}_n]_A \mid \Pi_{i=1}^{n'} \overline{x}\langle \vec{y}\rangle \mid [\Pi_{i=1}^{n''} \overline{x}\langle \vec{y}\rangle \mid \mathsf{R}]_B$$

PROOF: Define $\mathcal{R}$ by

$$(\nu \vec{c})(\mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{Q}) \mid \Pi_{i=1}^k \overline{a_i}\langle \vec{b_i}\rangle) \;\; \mathcal{R} \;\; (\nu \vec{c})(\mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{Q}) \mid \Pi_{i=1}^k \overline{a_i}\langle \vec{b_i}\rangle)$$

where

$$\mathsf{M}^{t'}_{kk'k''}(\mathsf{P},\mathsf{R}) = [\mathsf{P} \mid \mathsf{Q}^{t'}_k]_A \mid \Pi_{i=1}^{k'} \overline{x}\langle \vec{y}\rangle \mid [\Pi_{i=1}^{k''} \overline{x}\langle \vec{y}\rangle \mid \mathsf{R}]_B$$

and $k, k', k'' \geq 0$, $0 \leq t' \leq t$. Setting $\mathsf{M} = \mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{Q}) \mid \Pi_{i=1}^k \overline{a_i}\langle \vec{b_i}\rangle$, and $\mathsf{N} = \mathsf{N}^{t''}_{nn'n''}(\mathsf{P},\mathsf{Q}) \mid \Pi_{i=1}^k \overline{a_i}\langle \vec{b_i}\rangle$ We extend $\phi(\cdot)$ to natural numbers: $\phi(0) = 0, \phi(t+1) = t$. we have the following transitions, neglecting restrictions and bound outputs as before.

1. $\mathsf{M} \xrightarrow{a(\vec{b})}_a \mathsf{M} \mid \overline{a}\langle\vec{b}\rangle$. This is matched by $\mathsf{N} \xrightarrow{a(\vec{b})}_a \mathsf{N} \mid \overline{a}\langle\vec{b}\rangle$, regardless of whether $\overline{a}\langle\vec{b}\rangle = \overline{x}\langle\vec{y}\rangle$ or not.

2. $\mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \mathsf{M}^{\phi(t')}_{mm'm''}(\mathsf{P}',\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle$ because $\mathsf{P} \xrightarrow{\tau}_a \mathsf{P}'$. This is matched by $\mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \mathsf{M}^{\phi(t'')}_{nn'n''}(\mathsf{P}',\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle$.

3. $\mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}') \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle$ because $\mathsf{R} \xrightarrow{\tau}_a \mathsf{R}'$. This is matched by $\mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t''}_{nn'n''}(\mathsf{P}',\mathsf{R}') \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle$.

4. $\mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \mid \overline{a}\langle\vec{b}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t'}_{mm'm''}(\mathsf{P} \mid \overline{a}\langle\vec{b}\rangle,\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle$ where $a \in A$. This is matched by

$$\mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \mid \overline{a}\langle\vec{b}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t''}_{nn'n''}(\mathsf{P} \mid \overline{a}\langle\vec{b}\rangle,\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle.$$

5. $\mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \mid \overline{a}\langle\vec{b}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \mid \overline{a}\langle\vec{b}\rangle \mid \overline{a}\langle\vec{b}\rangle$ where $\overline{x}\langle\vec{y}\rangle \neq \overline{a}\langle\vec{b}\rangle$. Then

$$\mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \mid \overline{a}\langle\vec{b}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \mid \overline{a}\langle\vec{b}\rangle \mid \overline{a}\langle\vec{b}\rangle$$

is a matching transition.

6. $\mathsf{M}^{t'}_{mm'+1m''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle$, matched by the empty transition sequence.

7. $\mathsf{M}^{t'}_{mm'+1m''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t'}_{mm'+2m''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle$, also matched by the empty transition sequence.

8. $\mathsf{M}^{t'}_{mm'm''+1}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}') \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle$ because $\overline{x}\langle\vec{y}\rangle \mid \mathsf{R} \xrightarrow{\tau}_a \mathsf{R}'$. If $n'' > 0$, this is matched by

$$\mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\tau}_a \mathsf{M}^{t''}_{nn'n''-1}(\mathsf{P},\mathsf{R}') \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle.$$

If $n'' = 0$ we must work a bit harder. There are two cases. Here is the first.

$$\begin{aligned} \mathsf{M}^{t''}_{n+1n'0}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle &\xrightarrow{\tau}_a \mathsf{M}^{\phi(t'')}_{n1+n'0}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \\ &\xrightarrow{\tau}_a \mathsf{M}^{\phi(t'')}_{nn'1}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \\ &\xrightarrow{\tau}_a \mathsf{M}^{\phi(t'')}_{nn'0}(\mathsf{P},\mathsf{R}') \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \end{aligned}$$

The second case begins as follows.

$$\begin{aligned} \mathsf{M}^{t''}_{0n'0}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle &\underbrace{\xrightarrow{\tau}_a \cdots \xrightarrow{\tau}_a}_{t''} \mathsf{M}^0_{0n'0}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \\ &\xrightarrow{\tau}_a \qquad \mathsf{M}^t_{1n'0}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^k \overline{a_i}\langle\vec{b_i}\rangle \end{aligned}$$

Now we proceed as in the previous case.

9. $\mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle \mid \overline{a}\langle\vec{b}\rangle \xrightarrow{\overline{a}\langle\vec{b}\rangle}_a \mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle$. A matching transition is $\mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle \mid \overline{a}\langle\vec{b}\rangle \xrightarrow{\overline{a}\langle\vec{b}\rangle}_a \mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle$.

10. $\mathsf{M}^{t'}_{mm'+1m''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{M}^{t'}_{mm'm''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle$. If $n' > 0$ then

$$\mathsf{M}^{t''}_{nn'n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{M}^{t''}_{nn'-1n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle$$

matches. Otherwise we proceed as in (8): There are two cases. Here is the first.

$$\begin{array}{lll} \mathsf{M}^{t''}_{n+10n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle & \xrightarrow{\tau}_a & \mathsf{M}^{\phi(t'')}_{n1n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle \\ & \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a & \mathsf{M}^{\phi(t'')}_{n0n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle \end{array}$$

The second case begins as follows.

$$\begin{array}{lll} \mathsf{M}^{t''}_{00n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle & \underbrace{\xrightarrow{\tau}_a \cdots \xrightarrow{\tau}_a}_{t''} & \mathsf{M}^{0}_{00n''}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle \\ & \xrightarrow{\tau}_a & \mathsf{M}^{t}_{1n'0}(\mathsf{P},\mathsf{R}) \mid \Pi_{i=1}^{k}\overline{a_i}\langle\vec{b_i}\rangle \end{array}$$

Now we proceed as in the previous case.

□

Clearly (5.4) is a special case of this last lemma.

LEMMA 51 *Assume the following hold.*

- $m, m', m'', n, n', n'' \geq 0$,
- $0 \leq t', t'' \leq t$, $a \neq x$,
- P *timer-free and*
- $x \in B$.

With $\mathsf{Q}^{s}_{k} = \Pi_{i=1}^{k}\overline{x}\langle\vec{y}a\rangle \mid \mathsf{U}^{s}$ *we have*

$$[\mathsf{P} \mid \mathsf{Q}^{t'}_{m}]_A \mid \Pi_{i=1}^{m'}\overline{a} \mid [\Pi_{i=1}^{m''}\overline{a} \mid \mathsf{R}]_B \approx_a [\mathsf{P} \mid \mathsf{Q}^{t''}_{n}]_A \mid \Pi_{i=1}^{n'}\overline{a} \mid [\Pi_{i=1}^{n''}\overline{a} \mid \mathsf{R}]_B$$

PROOF: The proof is very similar to that of the previous lemma and hence omitted.

□

We have now also proven (5.6).

LEMMA 52 *If* P *is timer-free and* $a, b \notin \mathsf{fn}(\mathsf{P})$ *then*

$$[\mathsf{P} \mid !b.\mathsf{timer}^{t}(a,\overline{b}) \mid \Pi_{i=1}^{k}\overline{b} \oplus \mathsf{0} \mid \Pi_{i=1}^{l}\overline{b} \mid \Pi_{i=1}^{m}\mathsf{0} \mid \Pi_{i=1}^{n}\overline{a}]_A \approx_a [\mathsf{P}]_A$$

for all $l, m, n \geq 0$.

PROOF: The relation $\mathcal{R}$ is given by

$$[\mathsf{P} \mid !b.\mathsf{timer}^t(a,\overline{b}) \mid \Pi_{i=1}^{d}\mathsf{timer}^{t_d}(a,\overline{b}) \mid \Pi_{i=1}^{l}\overline{b} \mid \Pi_{i=1}^{m}\mathsf{0} \mid \Pi_{i=1}^{n}\overline{a}]_A \mid \Pi_{i=1}^{s}\overline{c_i}\langle\vec{d_i}\rangle$$
$$\mathcal{R}$$
$$[\mathsf{P} \mid \Pi_{i=1}^{l'}\overline{b} \mid \Pi_{i=1}^{n'}\overline{a}]_A \mid \Pi_{i=1}^{k}\overline{c_i}\langle\vec{d_i}\rangle$$

where $d, k, l, l', m, m', n \geq 0$ and P is timer-free with $a, b \notin \mathsf{fn}(\mathsf{P})$. We omit to show that $\mathcal{R} \cup \approx_a$ is indeed an asynchronous bisimulation, as that is straightforward. □

5.8 Related Work

Process theoretic studies of message failure are rare, but many authors have proposed and investigated process calculi intended for modelling certain features of DS. Most of that work features sites or locations in some form. In [26] one can find an overview. Almost all proposals for extending process calculi with sites have been made in the context of code mobility. This vogue is probably a consequence of Java's success, which took semanticists by surprise. For comparisons with π_{mlt} this is unfortunate, because code mobility is a powerful computational primitive: so much so in fact, that the Ambient Calculus [25] gets away with mobility as its sole means for computing. Adding an expressive mechanism to something as lightweight as name-passing interaction might result in the former masking subtle semantic effects of the latter.

Anyway, formalisms for message failure can be divided according to their mechanism for classifying interactions into those that are subject to failure and those that ain't. As we have seen, π_{mlt} uses sites to distinguish between local and remote communication. Abdulla and his collaborators propose to partition channels into those that carry messages, subject to possible failure, and those that don't [3, 4]. Of course finer distinctions are possible, for example by assigning probabilities.

Of the many formalisms using sites, if for other purposes, the following are most relevant here.

- Dπ, introduced by Hennessy and Riely [96–98].
- A proposal by Sewell: dpi [108].
- The Seal Calculus [117] by Vitek and Castagna.
- The aforementioned Ambient Calculus by Cardelli and Gordon.
- The Distributed Join Calculus by [39] by Gonthier, Lévy, Maranget and Rémy.
- And finally the earliest relevant model, π_l by Amadio and Prasad [8, 9].

With the exception of π_l all these formalisms have been proposed to investigate code mobility. Once one decides on using sites, several taxonomies become relevant.

- What is the underlying non-distributed calculus? Here we see much uniformity: the Distributed Join Calculus is based on the Join calculus and hence indirectly on a peculiar form of the asynchronous π-calculus. Dπ, dpi and π_l are based directly on some π-calculus. Ambient and Seal are not really based on existing calculi. Our π_{mlt} is unusual as it is the only formalism based on a timed calculus, which in our opinion, is crucial. Just as in the case of timers, we conjecture that expressivity differences in base calculi are preserved by the transition to their located incarnations.

- Can sites be nested or have they got to be flat? This pertains to the question of whether sites can themselves contain other sites or not. We have opted for the latter, as do Dπ and π_l. All the others have chosen to allow nesting of sites. This is probably right for dealing with code mobility's security issues, but as units of message failure, flat sites appear more natural, because in real computing systems, the unit of failure are physical devices such as PCs or routers that cannot meaningfully be nested.

- How are sites achieved? Our solution is primitive but effective. We added new syntax. Ambient, Seal, π_l and Dπ do the same. The guiding metaphor is spatial containment: a given process is executed 'inside' some computer. The Distributed Join Calculus and dpi make a different choice. They locate names. That means roughly that names are no longer atomic but structured, with two components, one of which represents the location. Maffeis and Carbone [24] study a pure form of structured names. We conjecture that these two approaches are not as different as may seem on first glance.

- Another interesting issue that could be seen as a special case of the choice of underlying calculus, is the choice of mechanism for failure detection. Our weapon here is the timer because it is powerful, natural and used by important distributed algorithms like TCP. Strangely, apart from π_l, none of the other formalisms has failure detection. π_l offers a dedicated primitive that can tell if a remote site has crashed or not. Since implementations of failure detectors tend to be built on top of timing mechanisms (but probably not vice versa), one could argue that our's is the more versatile and basic choice. Be that as it may, other detection technology must also be addressed. An important example are unique message IDs. Since the π-calculus offers unique name generation as a primitive, one could be tempted to think it was perfectly suited for modelling algorithms that rely on unique IDs. This is not the case. Firstly, many π-calculi do not allow to test names for (in)equality. It is

possible to code around this shortcoming, but at the expense of complicating whatever is being modelled. A more serious problem is the mismatch between π's ability to generate infinitely many new names and the finite number of IDs that a message can realistically carry in modern networks which has serious algorithmic ramifications. An extreme case is the Alternating Bit Protocol [111] that uses just one bit to generate message IDs for detecting message failures. The cleverness in such algorithms lays in how they get around the limited number of IDs. By assuming an infinite supply from the start, π-calculi cannot model such resource constrains well. It would be interesting to study a π-calculus that offers only a finite supply of fresh names. We imagine that the mathematical theory of such processes would be closely related to, and a generalisation of space-restricted Turing Machines, as studied in classical complexity theory.

5.9 Concluding Remarks

We have extended π_t with a notion of site and message failure to get a convenient model for the study of DS. Much could be added to increase its realism and in Chapter 7 we shall take a brief trip in that direction. Nevertheless, it might be more interesting at this stage, to avoid complicating an already challenging formalism to the point of infeasibility. Instead we suggest to study π_{mlt} more deeply, for example by getting more powerful expansion theorems or generalisations of the Recursive Timer. A thorough investigation is likely to show that there are only few ways of how to deal with message failures. It may be possible to capture some with suitable typing systems. We hope to be able to report of progress in this direction soon.

If one wishes to have a more realistic model, our suggestion is to start with making π_{mlt} less asynchronous. With π_t we had the opposite problem, we wanted more asynchrony, which means that we wanted π_t timers to exhibit some degree of clock-drift, cf. §4.7. Here it is the lack of synchronisation between timers in different sites that makes the model unrealistic. But with modern clock synchronisation algorithms [73] it is possible to push inter-site clock-drift below the average inter-site communication latency (which is still several orders of magnitude above the duration of atomic computational steps). It is likely that moving from a totally asynchronous model with no constraints on inter-site clock-drift to a partially synchronous one with some constraints would have serious semantic effects, for example a shift away from an impossibility to solve the leadership election problem across sites, that we presume π_{mlt} suffers from. The good news is that many of the techniques we believe could be used to make π_t less asynchronous are dual-use and would allow to make π_{mlt} more so (and vice versa).

Chapter 6

The 2PCP With Message Failure

This chapter studies the 2PCP assuming the possibility of message failure.

6.1 Introduction

We now have a distributed π-calculus at our disposal and this chapter tests it by encoding and verifying the 2PCP, first described in Chapter 3. Now participants and the coordinator reside in their own locations, which means that votes and decision messages can get lost or duplicated. But the algorithm in Chapter 3 works only in the absence of message failures. It would deadlock if a message got lost. So we scaffold the core algorithm with timers to enable the 2PCP to recover gracefully from message failure.

6.2 The Algorithm

As before, the protocol has n participants and one coordinator, all hosted by dedicated locations. Internal communication happens via the private names $\vec{vote}$, $\vec{dec}$, $\vec{e}$. The additional $\vec{e}$ is used for error recovery, as explained later. All processes carry an index, denoting the time units left until time-out of their timers.

$$\mathsf{2PCP}_{t,t'} = (\nu \vec{vote})(\nu \vec{dec})(\nu \vec{e})([\mathsf{C}_t]_{\vec{vote},\vec{e}} \mid [\mathsf{P}_{1,t'}]_{dec_1} \mid ... \mid [\mathsf{P}_{n,t'}]_{dec_n})$$

Participants are almost unchanged from the error free version, except that the receipt of the decision from the coordinator is wrapped in a form of Recursive

Timer, cf. §5.7.1.

$$\begin{aligned}
\mathsf{P}_{i,t} &= \mathsf{P}^{abort}_{i,t} \oplus \mathsf{P}^{commit}_{i,t} \\
\mathsf{P}^{abort}_{i,t} &= \overline{vote_i}\mathsf{right} \,|\, !\overline{abort_i} \\
\mathsf{P}^{commit}_{i,t} &= \overline{vote_i}\mathsf{left} \,|\, \mathsf{P}^{wait}_{i,t} \\
\mathsf{P}^{wait}_{i,t} &= \mathsf{timer}^t(dec_i[!\overline{commit_i}, !\overline{abort_i}], \overline{e_i} \,|\, \mathsf{P}^{wait}_{i,t})
\end{aligned}$$

If the decision message on dec_i does not arrive within t units of time, a time-out occurs, starting a new timer waiting for the decision and sending a message on e_i back to the coordinator. This message on e_i triggers a resend of the decision message. If the decision on dec_i fails every time, the 'process' made up of $\mathsf{P}^{wait}_{i,t}$ and its counterpart in the coordinator responsible for sending the decision diverge, but at each step in their joint evolution, the duo is capable of resuming error free service. This insensitivity to previous message failures is the reason behind the term "Recursive Timer".

The coordinator is also almost unchanged from its non-distributed incarnation, except that the receipt of a vote is now guarded by a timer. Its function is to make the coordinator assume a participant has voted to abort if no message to the contrary arrives in time. An additional new feature is that the sending of a decision message for the descendant of P_i can now be triggered by receipt of a message on e_i.

$$\begin{aligned}
\mathsf{C}_t &= \mathsf{C}^{abort} \oplus \mathsf{C}^{commit}_{pre,t} \\
\mathsf{C}^{abort} &= \Pi^n_{i=1}\mathsf{C}^{abort}_i \\
\mathsf{C}^{abort}_i &= \overline{dec_i}\mathsf{right} \,|\, !e_i.\overline{dec_i}\mathsf{right}
\end{aligned}$$

$$\begin{aligned}
\mathsf{C}^{commit}_{pre,t} &= (\nu\vec{c}a)(\mathsf{C}^{wait}_t \,|\, \mathsf{C}^{and} |\, \mathsf{C}^{or}) \\
\mathsf{C}^{wait}_t &= \Pi^n_{i=1}\mathsf{C}^{wait}_{i,t} \\
\mathsf{C}^{wait}_{i,t} &= \mathsf{timer}^t(vote_i[\overline{c_i}, \overline{a}], \overline{a}) \\
\mathsf{C}^{and} &= c_1.c_2...c_n.\mathsf{C}^{commit}_{final} \\
\mathsf{C}^{commit}_{final} &= \Pi^n_{i=1}\mathsf{C}^{commit}_i \\
\mathsf{C}^{commit}_i &= \overline{dec_i}\mathsf{left} \,|\, !e_i.\overline{dec_i}\mathsf{left} \\
\mathsf{C}^{or} &= a.\mathsf{C}^{abort}
\end{aligned}$$

Clearly $\mathsf{C}^{wait}_{i,t}$ makes worst case assumptions: it is possible for a participant to decide for voting to commit, without the vote arriving in time at the coordinator. The coordinator then incorrectly but safely assumes that the participant had voted to abort. This is a conservative assumption, but exactly the right one in an asynchronous system where it is never clear if a message has not arrived because it was lost or if it is just late.

From now on we will mostly omit specifying the access points of our networks. For example we will write $[\mathsf{C}_t]$ instead of $[\mathsf{C}_t]_{\vec{vote},\vec{e}}$ the formalisms more readable.

Remarks on the Underlying Calculus

Just like in Chapter 3, the calculus we have used here to express the 2PCP is not π_{mlt}, but a variant with binary branching, replicated outputs and definition of processes by recursive equations. As before, the reason for this choice is economy of presentation. It is no problem to show that all our results for π_{mlt} hold for the modified calculus, too. Unfortunately, and in contradistinction to the situation for the 2PCP without failures, it is *not* known if the above modifications can be encoded nicely into π_{mlt}.

6.3 Correctness of the 2PCP

As in the failure free case, a crucial part of what counts as correctness of the 2PCP is to do with appearing as being only in one of two states: either all processes abort or they all commit. This is guaranteed by the following theorem which is a located version of the Theorems 12 in Chapter 3.

THEOREM 33 *Let $t, t' > 0$, then $\mathsf{2PCP}_{t,t'} \approx_a [\mathsf{Abort} \oplus \mathsf{Commit}]_\emptyset$.*

In Chapter 3 we added Theorem 13 to guarantee that the overall outcome of the protocol was the conjunction of the $n+1$ individual decisions. With message failures this is no longer possible: even if all votes had been cast for committing, a lost vote would force the coordinator to abort. The following modification of Theorem 13 takes this into account.

THEOREM 34 *1.* $(\nu \vec{vote})(\nu \vec{dec})(\nu \vec{e})([\mathsf{C}^{commit}_{pre,t}] \mid [\mathsf{P}^c_{1,t}] \mid ... \mid [\mathsf{P}^c_{n,t}]) \approx_a [\mathsf{Commit} \oplus \mathsf{Abort}]_\emptyset$.

2. If $\mathsf{P}'_i \in \{\mathsf{P}^a_{i,t}, \mathsf{P}^c_{i,t}\}$, *then* $(\nu \vec{vote})(\nu \vec{dec})(\nu \vec{e})([\mathsf{C}^{abort}] \mid [\mathsf{P}'_1] \mid ... \mid [\mathsf{P}'_n]) \approx_a [\mathsf{Abort}]_\emptyset$.

3. If $\mathsf{P}'_i \in \{\mathsf{P}^a_{i,t}, \mathsf{P}^c_{i,t}\}$ *and for at least one* $i_0 \in \{1, ..., n\}$ $\mathsf{P}'_{i_0} = \mathsf{P}^a_{i_0,t}$, *then*

$$(\nu \vec{vote})(\nu \vec{dec})(\nu \vec{e})([\mathsf{C}^{commit}_{pre,t}] \mid [\mathsf{P}'_1] \mid ... \mid [\mathsf{P}'_n]) \approx_a [\mathsf{Abort}]_\emptyset.$$

The caveats discussed in Chapter 3 apply here too: Theorems 33 and 34 do not exhaust all that one could say about the correctness of the 2PCP. However, unlike in the failure free case where the missing bits were trivial, the possibility of message failure in intersite communication requires an additional guarantee, usually informally expressed as "if [...] no [...] failure occurs for sufficiently long, then all processes will eventually reach a decision" [19]. This pertains to the possibility of messages getting lost *every time* they leave their originating site. Since a participant who has voted to commit cannot consistently decide on whether to abort or

commit without feedback from the coordinator, at least one message to each such participant has to get through for the protocol to be atomic. Unfortunately, our semantic model does not easily accommodate quantitative information about message failure frequency. This makes it inadvisable to prove theorems about computations that put limits on the number of message failures. Does that mean we have no way of expressing the robustness of the 2PCP to limited numbers of failures? This is not clear. We will prove Theorem 33 and it implies that no observer could ever see anything other than total commitment or overall abort from the 2PCP, despite the aforementioned possibility of a participant being blocked because the coordinator's decision will never arrive. This seems contradictory and it is not entirely clear how to explain this. It seems that at the heart of any solution to this conundrum lays the *divergence insensitivity* of the relevant equivalences. This phrase is usually taken to mean

$$\mathcal{T}_{max} \vdash \mathsf{P} = \mathsf{P} \,|\, \Omega \tag{6.1}$$

(in the case of the maximal sound theory) where Ω is a divergent and insensitive process. Clearly, infinitely iterated message failure is a form of divergence. But that cannot be the whole story here, for we could also have a

$$\mathsf{2PCP} \twoheadrightarrow \mathsf{2PCP}' \xrightarrow{\overline{abort_1}} \xrightarrow{\tau} \ldots \xrightarrow{\tau} \xrightarrow{\overline{abort_1}} \xrightarrow{\tau} \ldots \xrightarrow{\tau} \xrightarrow{\overline{abort_1}} \xrightarrow{\tau} \ldots \xrightarrow{\tau} \ldots$$

where all the τ-actions $\xrightarrow{\tau} \ldots \xrightarrow{\tau}$ are either losses of $\overline{dec_2}\mathsf{right}$, time steps by the second participant or triggers to resend the lost message. This means we have a proper subprocess diverging, but not being insensitive, at least from the point of view of the coordinator. In addition, the divergence is interleaved with observable actions. Is seems that our equivalences are divergence insensitive in a stronger sense than that expressed in (6.1). It appears crucial that the diverging subprocess is oblivious about previous message failure and may return to non-faulty behaviour at each point in its evolution. As it is more important to know *that* $\mathsf{2PCP} \approx_a [\mathsf{Abort} \oplus \mathsf{Commit}]_\emptyset$ than *why*, we leave a better understanding of just what kind of divergence insensitivity the 2PCP needs and $\approx_a$ offers as an open problem.

6.4 Correctness of the 2PCP with One Participant

The first correctness proof of the 2PCP did not constrain the number of participants. Here we are less ambitious and only establish correctness for the protocol with one participant. Just 1 participant? Is there a problem? Yes and no! No because it is not conceptually difficult to generalise the proof we are about to present to n participants. Unfortunately, if the coordinator has to communicate with more than one process, the number of intermediate states, induced by messages in transit, increases dramatically. None of these additional states is challenging in itself, it is

their sheer number that makes simply keeping track a formidable task. A quick look at Lemma 57 may be instructive: despite stating a fairly straightforward equality, its proof uses a process with 13 indices! With more participants things become much worse.

Is this complexity inevitable? Probably, but ... We explored quite a few variants of the proof and all suffered from an explosion of intermediate states. We felt like pushing a bump in the carpet: we could move complexity around but failed to remove it. On positive side, one may notice a fair degree of regularity in the intermediate state bureaucracy, raising the spectre of a general theorem. Sadly, its details, unlike its spirit, have resisted the author's considerable efforts so far.

Some Useful Definitions and Lemmas

We begin the proof with a brief account of S-bisimulations, which is very similar to what we did in Chapter 3, except that this time we deal with networks, not processes.

DEFINITION 67 Let S be a set of names. We call actions l with $\mathsf{fn}(l) \cap S \neq \emptyset$ *S-hidden*. A network is *static* if all of its or its descendants outputs are free outputs.

A binary relation $\mathcal{R}$ on networks is an *asynchronous S-bisimulation* if $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ implies that P as well as Q are static and

- whenever $\mathsf{P} \stackrel{l}{\longrightarrow}_a \mathsf{P}'$ and $\mathsf{fn}(l) \cap S = \emptyset$, then $\mathsf{Q} \stackrel{\hat{l}}{\Longrightarrow}_a \mathsf{Q}'$ for some Q' such that $(\mathsf{P}', \mathsf{Q}') \in \mathcal{R}$,
- and vice versa.

By $\approx_a^S$ we denote the largest asynchronous S-bisimulation. The definitions of *strong asynchronous S-bisimulation* and $\sim_a^S$ are similar.

The next two theorems are adaptations of Theorems 14 and 15. They explain how the various equivalences relate and give some of their closure properties.

THEOREM 35 $\sim_a^S \subseteq \approx_a^S$.

PROOF: Straightforward. □

THEOREM 36 *Let $\mathcal{R}^S$ be one of $\approx_a^S$ and $\sim_a^S$.*

1. *$\mathsf{M}\,\mathcal{R}^S\,\mathsf{N}$ implies $(\nu x)\mathsf{M}\,\mathcal{R}^{S\setminus\{x\}}\,(\nu x)\mathsf{N}$ and $\mathcal{R}^\emptyset \subseteq \mathcal{R}$.*
2. *If $S \subseteq T$, then $\mathcal{R}^S \subseteq \mathcal{R}^T$.*
3. *If $\mathsf{M}\,\mathcal{R}^S\,\mathsf{N}$ and $\mathsf{fn}(\mathsf{R}) \cap S = \emptyset$ then $\mathsf{M} \mid \mathsf{L}\,\mathcal{R}^S\,\mathsf{N} \mid \mathsf{L}$.*

PROOF: Straightforward. □

The next lemma shows some elementary properties of the internal sum.

LEMMA 53 *1.* $\mathsf{P} \oplus \mathsf{P} \approx_a \mathsf{P}$.

2. Assume that $x, y \notin \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q})$. *Then*

$$(\nu xy)(\overline{x} \oplus \overline{y} \mid x.\mathsf{P} \mid y.\mathsf{Q}) \approx_a \mathsf{P} \oplus \mathsf{Q}$$

PROOF: Straightforward. □

The following lemma corresponds to Lemma 3 in Chapter 3.

LEMMA 54 *If* $x \in H \setminus (A \cup \mathsf{fn}(\mathsf{M}) \cup \mathsf{fn}(\mathsf{P}))$, *then*

1. $\mathsf{M} \mid [\overline{x}\mathsf{right}\langle\vec{y}\rangle \mid \mathsf{P}]_A \approx_a^H \mathsf{M} \mid [\mathsf{P}]_A$
2. $\mathsf{M} \mid [\overline{x}\mathsf{left}\langle\vec{y}\rangle \mid \mathsf{P}]_A \approx_a^H \mathsf{M} \mid [\mathsf{P}]_A$
3. $\mathsf{M} \mid [\overline{x}\langle\vec{y}\rangle \mid \mathsf{P}]_A \approx_a^H \mathsf{M} \mid [\mathsf{P}]_A$
4. $\mathsf{M} \mid [x[(\vec{v}).\mathsf{Q} \;\&\; (\vec{w}).\mathsf{R}] \mid \mathsf{P}]_A \approx_a^H \mathsf{M} \mid [\mathsf{P}]_A$
5. $\mathsf{M} \mid [x(\vec{v}).\mathsf{Q} \mid \mathsf{P}]_A \approx_a^H \mathsf{M} \mid [\mathsf{P}]_A$

PROOF: Immediate by assumptions. □

Next we touch on a crucial difference between $\approx_a$ and $\approx_a^H$: the latter can relate networks with different access points.

LEMMA 55 *1. Let* $\mathsf{M} \approx_a^H \mathsf{N}$. *If* $x \in \mathsf{ap}(\mathsf{M}) \setminus \mathsf{ap}(\mathsf{N})$ *or* $x \in \mathsf{ap}(\mathsf{N}) \setminus \mathsf{ap}(\mathsf{M})$, *then* $x \in H$.

2. If $A \subseteq H$ *and* $\mathsf{ap}(\mathsf{M}) \cup A = \emptyset$, *then* $[\mathsf{0}]_A \mid \mathsf{M} \approx_a^H \mathsf{M}$.

PROOF: For (1), assume $x \in \mathsf{ap}(\mathsf{M}) \setminus \mathsf{ap}(\mathsf{N})$, but $x \notin H$. Let $\overline{x}\langle\vec{y}\rangle$ not be H-hidden. Then

$$\mathsf{M} \xrightarrow{x(\vec{y})}_a \overline{x}\langle\vec{y}\rangle \mid \mathsf{M} \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{M}$$

But by Lemma 46.1, whenever

$$\mathsf{N} \twoheadrightarrow_a \mathsf{N}' \xrightarrow{x(\vec{y})}_a \overline{x}\langle\vec{y}\rangle \mid \mathsf{N}' \twoheadrightarrow_a \mathsf{N}'',$$

then $x \in \mathsf{ap}(\mathsf{N}'')$, so there cannot be a transition $\mathsf{N}'' \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{N}'''$, which means that M and N cannot be related by $\approx_a$.

For (2), we define $\mathcal{R}$ up to $\equiv$ by

$$\mathsf{M} \mid [\Pi_{i \in I} \overline{x_i}\langle\vec{y_i}\rangle]_A \;\; \mathcal{R} \;\; \mathsf{M}'$$

whenever $A \subseteq H$, $\mathsf{ap}(\mathsf{M}) \cap A = \emptyset$, $\mathsf{M} \equiv \mathsf{M}' \mid \Pi_{j \in J} \overline{x_j}\langle\vec{y_j}\rangle$ and $x_i \in A$ for all $i \in I \cup J$. Now $\mathsf{M} \mid [\Pi_{i \in I} \overline{x_i}\langle\vec{y_i}\rangle]_A$ has the following non H-hidden transitions.

- $\mathsf{M} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A \xrightarrow{l}_a \mathsf{N} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A$ because $\mathsf{M}' \xrightarrow{l}_a \mathsf{N}$, then $\mathsf{M}' \xrightarrow{l}_a \mathsf{N}$ is a matching transition.
- If $[\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A \xrightarrow{\tau}_a \mathsf{N}$, then the empty transition sequence matches
- If $\mathsf{M} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A \xrightarrow{x(\vec{y})}_a \mathsf{M} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A \mid \overline{x}\langle\vec{y}\rangle$, then $\mathsf{M}' \xrightarrow{x(\vec{y})}_a \mathsf{M}' \mid \overline{x}\langle\vec{y}\rangle$ matches because $x \notin A$.
- If $\mathsf{M} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A \mid \overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{M} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A$, then $x \notin A$ and consequently $\mathsf{M}' \mid \overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{M}'$ is a match.
- If $x \in A$ then $\mathsf{M} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A \xrightarrow{\tau}_a \mathsf{M} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle \mid \overline{x}\langle\vec{y}\rangle]_A$ can be matched by the empty transition sequence at M'.

The non H-hidden transitions of M' can obviously all be matched by $\mathsf{M} \mid [\Pi_{i\in I}\overline{x_i}\langle\vec{y_i}\rangle]_A$. □

Now we demonstrate how intersite communication on hidden names reduces to internal non-determinism.

LEMMA 56 *Assume that* R *and* S *are timer-free and* $x \in A \cup H$ *but* $x \notin \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q}) \cup \mathsf{fn}(\mathsf{R}) \cup \mathsf{fn}(\mathsf{S})$. *Then*

1. $[\mathsf{timer}^t(x[(\vec{v}).\mathsf{P} \,\&\, ().\mathsf{Q}], \mathsf{Q}) \mid \mathsf{R}]_A \mid [\overline{x}\mathsf{left}\langle\vec{y}\rangle \mid \mathsf{S}]_B \approx_a^H [\mathsf{P}\{\vec{y}/\vec{v}\} \oplus \mathsf{Q}]_A \mid [\mathsf{S}]_B$.
2. $[\mathsf{timer}^t(x[(\vec{v}).\mathsf{P} \,\&\, ().\mathsf{Q}], \mathsf{Q}) \mid \mathsf{R}]_A \mid [\overline{x}\mathsf{right} \mid \mathsf{S}]_B \approx_a^H [\mathsf{Q}]_A \mid [\mathsf{S}]_B$.
3. $[\mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \mid \mathsf{R}]_A \mid [\overline{x}\langle\vec{y}\rangle \mid \mathsf{S}]_B \approx_a^H [\mathsf{P}\{\vec{y}/\vec{v}\} \oplus \mathsf{Q}]_A \mid [\mathsf{S}]_B$.

PROOF: This is essentially the first Expansion Theorem 31. □

The following Lemma is an instance of the Recursive Timer.

LEMMA 57 *Let* $\mathsf{P}_t = \mathsf{timer}^t(x[(\vec{v}).\mathsf{Q} \,\&\, (\vec{w}).\mathsf{R}], \overline{e} \mid \mathsf{P}_t)$ *where* x, e *are fresh. Assume that* $k, l, m \geq 0$. *Then*

$$[\overline{x}\mathsf{right}\langle\vec{y}\rangle \mid !e.\overline{x}\mathsf{right}\langle\vec{y}\rangle \mid \Pi_{i=1}^k \overline{e}]_A \mid \Pi_{i=1}^l \overline{e} \mid [\Pi_{i=1}^m \overline{e} \mid \mathsf{P}_t]_B \approx_a^{\{x,e\}} [\mathsf{R}\{\vec{y}/\vec{w}\}]_B.$$

PROOF: Define

$$\begin{aligned}
\mathsf{A}_{I,I_l,I_r,J,K} &= \Pi_{i\in I}\overline{a_i}\langle\vec{b_i}\rangle \mid \Pi_{i\in I_l}\overline{a_i}\mathsf{left}\langle\vec{b_i}\rangle \mid \Pi_{i\in I_r}\overline{a_i}\mathsf{right}\langle\vec{b_i}\rangle \mid \Pi_{i\in J}\overline{e} \mid \Pi_{i\in K}\overline{x}\mathsf{right}\langle\vec{y}\rangle \\
\mathsf{L}^{t'}_{I_1,\ldots,I_{12}} &= [\mathsf{A}_{\emptyset,\emptyset,\emptyset,I_1,I_2} \mid \overline{x}\mathsf{right}\langle\vec{y}\rangle \mid !e.\overline{x}\mathsf{right}\langle\vec{y}\rangle]_{\{e\}} \mid \mathsf{A}_{I_3,I_4,I_5,I_6,I_7} \mid [\mathsf{A}_{I_8,I_9,I_{10},I_{11},I_{12}} \mid \mathsf{P}_{t'}]_B
\end{aligned}$$

Now define $\mathcal{R}$ up to $\equiv$ by

$$L^{t'}_{I_1,\ldots,I_{12}} \quad \mathcal{R} \quad [\mathsf{A}_{\emptyset,\emptyset,\emptyset,I_1,I_2}] \mid \mathsf{A}_{I_3,I_4,I_5,I_6} \mid [\mathsf{A}_{I_7,I_8,I_9,I_{10},I_{11},I_{12}}]_B$$

where $0 < t' \leq t$ and all the index sets I_j are chosen appropriately, e.g. for all $i \in I_7 : \overline{a_i}\langle\vec{b_i}\rangle \in B$, $\overline{a_i}\langle\vec{b_i}\rangle \neq \overline{e}$ etc. It is straightforward but rather labourious to show that $\mathcal{R}$ is an asynchronous $\{x, e\}$-bisimulation. □

Next is the Divide-and-Conquer Theorem. It is easier than that of Chapter 3 because we need it only for situations with one participant.

THEOREM 37 *1. Let I, J be two index sets such that for all $i \in I$ and $j \in J$, $\mathsf{M}_i, \mathsf{N}_i$ are networks and $\mathsf{P}_i, \mathsf{Q}_j$ are processes such that*

- $\mathsf{fn}(\mathsf{M}_i) \subseteq H$ *and* $A, B, \mathsf{ap}(\mathsf{M}_i) \subseteq H$*;*
- $\mathsf{M}_i \mid [\mathsf{P}_i \mid \mathsf{Q}_j]_A \approx_a^H \mathsf{N}_i$*;*
- $\mathsf{M}_i \mid [\mathsf{P}_i \mid \mathsf{R}_j]_A \approx_a^H \mathsf{N}_i$*; and*
- *whenever* $\mathsf{M}_i \mid [\mathsf{P}_i]_A \xrightarrow{l}_a \mathsf{M}$ *and l is neither H-hidden or an input, then for some $k \in I$:* $\mathsf{M} \equiv \mathsf{M}_k \mid [\mathsf{P}_k]_A$ *and* $\mathsf{N}_i \xrightarrow{\hat{l}}_a \mathsf{N}_k$.

Then $\mathsf{M}_i \mid [\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_j)]_A \approx_a^H \mathsf{N}_i$.

2. Let $\{\mathsf{M}_i\}_{i \in I}$ be a collection of networks and assume $\mathsf{P}, \mathsf{Q}, \mathsf{R}, \mathsf{S}$ are processes such that for all $i \in I$:

- $\mathsf{fn}(\mathsf{M}_i) \subseteq H$ *and* $A, B, \mathsf{ap}(\mathsf{M}_i) \subseteq H$*;*
- $\mathsf{M}_i \mid [\mathsf{P}]_A \approx_a^H [\mathsf{R} \oplus \mathsf{S}]_B$*;*
- $\mathsf{M}_i \mid [\mathsf{Q}]_A \approx_a^H [\mathsf{R}]_B$*; and*
- *whenever* $\mathsf{M}_i \xrightarrow{l}_a \mathsf{M}$ *and l is not H-hidden then at least one of the following must be true.*
  - $\mathsf{M} \mid [\mathsf{P} \oplus \mathsf{Q}]_A \approx_a^H [\mathsf{R}]_B$ *or*
  - $\mathsf{M} \equiv \mathsf{M}_j$ *for some $j \in I$.*

Then for all $i \in I$: $\mathsf{M} \mid [\mathsf{P} \oplus \mathsf{Q}]_A \approx_a^H [\mathsf{R} \oplus \mathsf{S}]_B$.

PROOF: Define $\mathcal{R}$ up to $\equiv$ by

$$\mathsf{M}_i \mid [\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_j)]_A \;\; \mathcal{R} \;\; \mathsf{N}_i$$

for all $i \in I$ and $j \in J$. We show that $\mathcal{R}$ is an asynchronous H-bisimulation up to $\equiv$.

Since by assumption, a transition $\mathsf{N}_i \xrightarrow{l}_a \mathsf{N}$ where l is not H-hidden, is matched my $\mathsf{M}_i \mid [\mathsf{P}_i \mid \mathsf{Q}_j]_A \xrightarrow{\hat{l}}_a \mathsf{M} \approx_a^H \mathsf{N}$, the transition

$$\mathsf{N}_i \xrightarrow{l}_a \mathsf{N}$$

where l is not H-hidden, is matched by

$$\mathsf{M}_i \mid [\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_j)]_A \xrightarrow{\tau}_a \mathsf{M}_i \mid [\mathsf{P}_i \mid \mathsf{Q}_j]_A \overset{\hat{l}}{\twoheadrightarrow}_a \mathsf{M}.$$

Conversely, $\mathsf{M}_i \mid [\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_j)]_A \xrightarrow{\tau}_a \mathsf{M}_i \mid [\mathsf{P}_i \mid \mathsf{Q}_j]_A$ and $\mathsf{M}_i \mid [\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_j)]_A \xrightarrow{\tau}_a \mathsf{M}_i \mid [\mathsf{P}_i \mid \mathsf{R}_j]_A$ are both matched by the empty transition sequence at N_i. Lastly, if

$$\mathsf{M}_i \mid [\mathsf{P}_i \mid (\mathsf{Q}_j \oplus \mathsf{R}_j)]_A \xrightarrow{l}_a \mathsf{M}' \mid [\mathsf{P}' \mid (\mathsf{Q}_j \oplus \mathsf{R}_j)]_A$$

where l is not H-hidden, then by assumption

$$\mathsf{M}' \mid [\mathsf{P}']_A \equiv \mathsf{M}_k \mid [\mathsf{P}_k]_A$$

and

$$\mathsf{N}_i \overset{\hat{l}}{\twoheadrightarrow}_a \mathsf{N}_k,$$

as required.

For (2) we proceed similarly. Define $\mathcal{R}$ (up to $\equiv$) by

$$\mathsf{M}_i \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{P} \oplus \mathsf{Q}]_B \quad \mathcal{R} \quad \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{R} \oplus \mathsf{S}]_B$$

where $i \in I$, $\{a_j, \vec{b_j}\} \subseteq H$ for all appropriate j. Then $\mathcal{R} \cup \approx_a^H$ is an asynchronous H-bisimulation. Let M abbreviate $\mathsf{M}_i \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{P} \oplus \mathsf{Q}]_B$ and N stand for $\Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{R} \oplus \mathsf{S}]_B$.

- $\mathsf{M} \xrightarrow{x(\vec{y})}_a \mathsf{M}_i \mid \mathsf{M} \mid \overline{x}\langle \vec{y} \rangle$, where $x(\vec{y})$ is not H-hidden, is matched by $\mathsf{N} \xrightarrow{x(\vec{y})}_a \mathsf{N} \mid \overline{x}\langle \vec{y} \rangle$.
- A message loss $\mathsf{M} \xrightarrow{\tau}_a \mathsf{M}_i \mid \Pi_{j=1, j \neq j_0}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{P} \oplus \mathsf{Q}]_B$ is matched by

  $$\mathsf{N} \xrightarrow{\tau}_a \Pi_{j=1, j \neq j_0}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{R} \oplus \mathsf{S}]_B.$$

- A message duplication $\mathsf{M} \xrightarrow{\tau}_a \mathsf{M}_i \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid \overline{a_{j_0}}\langle \vec{b_{j_0}} \rangle \mid [\mathsf{P} \oplus \mathsf{Q}]_B$ is matched by $\mathsf{N} \xrightarrow{\tau}_a \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid \overline{a_{j_0}}\langle \vec{b_{j_0}} \rangle \mid [\mathsf{R} \oplus \mathsf{S}]_B$.
- A decision $\mathsf{M} \xrightarrow{\tau}_a \mathsf{M}_i \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{P}]_B$ is matched by $\mathsf{N} \xrightarrow{\tau}_a \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{R} \oplus \mathsf{S}]_B$. The corresponding decision for Q is matched similarly.
- If $\mathsf{M} \xrightarrow{\tau}_a \mathsf{M}' \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{P} \oplus \mathsf{Q}]_B$ we have two possibilities. If $\mathsf{M}' \mid [\mathsf{P} \oplus \mathsf{Q}]_A \approx_a^H [\mathsf{R}]_B$, then also $\mathsf{M}' \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \mid [\mathsf{P} \oplus \mathsf{Q}]_A \approx_a^H [\mathsf{R}]_B \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle$, so

  $$[\mathsf{R} \oplus \mathsf{S}]_B \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle \xrightarrow{\tau}_a [\mathsf{R}]_B \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle$$

  is a match. Otherwise $\mathsf{M}' \equiv \mathsf{M}_j$ for some $j \in I$ and the empty transition sequence at $[\mathsf{R} \oplus \mathsf{S}]_B \mid \Pi_{j=1}^m \overline{a_j}\langle \vec{b_j} \rangle$ matches.

□

The Core of the Proof

For convenience we present the processes that make up the 2PCP with one participant before we go to the guts of the correctness proof. We start with the overall protocol.

$$\mathsf{2PCP}_t \;=\; (\nu vote)(\nu dec)(\nu e)(\mathsf{C}_t \mid \mathsf{P})$$

Next is the coordinator.

$$\begin{array}{rcl}
\mathsf{C}_t & = & [\mathsf{C}^{abort} \oplus \mathsf{C}^{commit}_{pre,t}] \\
\mathsf{C}^{abort} & = & \overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right} \\
\mathsf{C}^{commit}_{pre,t} & = & (\nu ca)(\mathsf{timer}^t(vote[\overline{c},\overline{a}],\overline{a}) \mid c.\mathsf{C}^{commit} \mid a.\mathsf{C}^{abort}) \\
\mathsf{C}^{commit} & = & \overline{dec}\mathsf{left} \mid !e.\overline{dec}\mathsf{left}
\end{array}$$

Finally, teh sole participant.

$$\begin{array}{rcl}
\mathsf{P} & = & [\mathsf{P}^a \oplus \mathsf{P}^c_t] \\
\mathsf{P}^a & = & \overline{vote}\mathsf{right} \mid !\overline{abort} \\
\mathsf{P}^c_t & = & \overline{vote}\mathsf{left} \mid \mathsf{P}'_t \\
\mathsf{P}'_t & = & \mathsf{timer}^t(dec[!\overline{commit}, !\overline{abort}], \overline{e} \mid \mathsf{P}'_t)
\end{array}$$

Definition 68 The set of restricted hidden names that we use below is $H = \{vote, dec, e\}$.

The next lemma corresponds to Lemma 4 and states that the coordinator's abort is sufficient for overall abort.

Lemma 58 *Let $t > 0$.*

1. $[\mathsf{C}^{abort}] \mid [\mathsf{P}^a] \approx^H_a [!\overline{\mathrm{abort}}]$.
2. $[\mathsf{C}^{abort}] \mid [\mathsf{P}^c] \approx^H_a [!\overline{\mathrm{abort}}]$.
3. $[\mathsf{C}^{abort}] \mid [\mathsf{P}] \approx^H_a [!\overline{\mathrm{abort}}]$.

Proof: We begin by deriving (1).

$$\begin{array}{rcll}
[\mathsf{C}^{abort}] \mid [\mathsf{P}^a] & \equiv & [\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}] \mid [\overline{vote}\mathsf{right} \mid !\overline{abort}] & \\
& \approx^H_a & [\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}] \mid [!\overline{abort}] & (6.2) \\
& \approx^H_a & [\mathsf{0}] \mid [!\overline{abort}] & (6.3) \\
& \approx^H_a & [!\overline{abort}] & (6.4)
\end{array}$$

For (2), we proceed similarly.

$$\begin{aligned}
[\mathsf{C}^{abort}] \,|\, [\mathsf{P}^c_t] &\equiv [\overline{dec}\mathsf{right} \,|\, !e.\overline{dec}\mathsf{right}] \,|\, [\overline{vote}\mathsf{left} \,|\, \mathsf{P}'_t] \\
&\approx^H_a [\mathsf{0}] \,|\, [\overline{vote}\mathsf{left} \,|\, !\overline{abort}] && (6.5)\\
&\approx^H_a [\mathsf{0}] \,|\, [!\overline{abort}] && (6.6)\\
&\approx^H_a [!\overline{abort}] && (6.7)
\end{aligned}$$

Now the only non H-hidden transition C^{abort} can engage in, is the idle step $\mathsf{C}^{abort} \xrightarrow{\tau}_a \mathsf{C}^{abort}$, so we can apply Theorem 37.1 to infer (3) from (1) and (2). □

Next is the equivalent of Lemma 5.

LEMMA 59 *Let $t > 0, t' > 0$.*

1. $[\mathsf{C}^{commit}_{pre,t}] \,|\, [\mathsf{P}^a] \approx^H_a [!\overline{\text{abort}}]$.
2. $[\mathsf{C}^{commit}_{pre,t}] \,|\, [\mathsf{P}^c_{t'}] \approx^H_a [!\overline{\text{abort}} \oplus !\overline{\text{commit}}]$.
3. $[\mathsf{C}^{commit}_{pre,t}] \,|\, [\mathsf{P}_{t'}] \approx^H_a [!\overline{\text{abort}} \oplus !\overline{\text{commit}}]$.

PROOF: The proof of (1) is a straightforward equation.

$$\begin{aligned}
[\mathsf{C}^{commit}_{pre,t}] \,|\, [\mathsf{P}^a] &\equiv [(\nu ca)(\mathsf{timer}^t(vote[\overline{c},\overline{a}],\overline{a}) \,|\, c.\mathsf{C}^{commit} \,|\, a.\mathsf{C}^{abort})] \,|\, [\overline{vote}\mathsf{right} \,|\, !\overline{abort}] \\
&\approx^H_a [(\nu ca)(\overline{a} \oplus \overline{a} \,|\, c.\mathsf{C}^{commit} \,|\, a.\mathsf{C}^{abort})] \,|\, [!\overline{abort}] && (6.8)\\
&\approx^H_a [(\nu ca)(\overline{a} \,|\, c.\mathsf{C}^{commit} \,|\, a.\mathsf{C}^{abort})] \,|\, [!\overline{abort}] && (6.9)\\
&\approx^H_a [(\nu a)(\overline{a} \,|\, a.\mathsf{C}^{abort})] \,|\, [!\overline{abort}] && (6.10)\\
&\approx^H_a [\mathsf{C}^{abort}] \,|\, [!\overline{abort}] && (6.11)\\
&\equiv [\overline{dec}\mathsf{right}] \,|\, [!\overline{abort}] && (6.12)\\
&\approx^H_a [\mathsf{0}] \,|\, [!\overline{abort}] && (6.13)\\
&\approx^H_a [!\overline{abort}] && (6.14)
\end{aligned}$$

For (2), we must work a bit harder.

$$\begin{aligned}
[\mathsf{C}^{commit}_{pre,t}] \,|\, [\mathsf{P}^c_{t'}] &\equiv [(\nu ca)(\mathsf{timer}^t(vote[\overline{c},\overline{a}],\overline{a}) \,|\, c.\mathsf{C}^{commit} \,|\, a.\mathsf{C}^{abort})] \,|\, [\overline{vote}\mathsf{left} \,|\, \mathsf{P}'_{t'}] \\
&\approx^H_a [(\nu ca)(\overline{c} \oplus \overline{a} \,|\, c.\mathsf{C}^{commit} \,|\, a.\mathsf{C}^{abort})] \,|\, [\mathsf{P}'_{t'}] && (6.15)
\end{aligned}$$

Now for all $k, l, m \geq 0$

$$\begin{aligned}
[\mathsf{C}^{abort} \,|\, \Pi^k_{i=1}\overline{e}] \,|\, \Pi^l_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^m_{i=1}\overline{e}] &\approx^H_a [\mathsf{0}] \,|\, [!\overline{abort}] && (6.16)\\
&\approx^H_a [!\overline{abort}] && (6.17)
\end{aligned}$$

$$\begin{aligned}
[\mathsf{C}^{commit} \,|\, \Pi^k_{i=1}\overline{e}] \,|\, \Pi^l_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^m_{i=1}\overline{e}] &\approx^H_a [\mathsf{0}] \,|\, [!\overline{commit}] && (6.18)\\
&\approx^H_a [!\overline{commit}] && (6.19)
\end{aligned}$$

This together with Theorem 37.2 implies

$$[\mathsf{C}^{abort} \oplus \mathsf{C}^{commit}_{pre,t}] \,|\, [\mathsf{P}^{c}_{t'}] \approx^{H}_{a} [!\overline{abort} \oplus !\overline{commit}]$$

because $[\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}]$ has exactly the following non H-hidden, non-input transitions.

- $[\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'+1} \,|\, \Pi^{m}_{i=1}\overline{e}] \xrightarrow{\tau}_{a} [\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}]$, an application of (IDLE) to the process in the right network.
- $[\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{1} \,|\, \Pi^{m}_{i=1}\overline{e}] \xrightarrow{\tau}_{a} [\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m+1}_{i=1}\overline{e}]$, a time-out in the right network.
- $[\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}] \xrightarrow{\tau}_{a} [\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}]$, an (IDLE) in the network on the left.
- $[\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m+1}_{i=1}\overline{e}] \xrightarrow{\tau}_{a} [\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l+1}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}]$, a message $\overline{e}$ migrating from its originating site to the ether.
- $[\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l+1}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}] \xrightarrow{\tau}_{a} [\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}]$, the loss of one $\overline{e}$.
- $[\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l+1}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}] \xrightarrow{\tau}_{a} [\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l+2}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}]$, the duplication of one $\overline{e}$.
- $[\Pi^{k}_{i=1}\overline{e}] \,|\, \Pi^{l+1}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}] \xrightarrow{\tau}_{a} [\Pi^{k+1}_{i=1}\overline{e}] \,|\, \Pi^{l}_{i=1}\overline{e} \,|\, [\mathsf{P}'_{t'} \,|\, \Pi^{m}_{i=1}\overline{e}]$, the migration of one $\overline{e}$ to its target site.

All of them are trivially matched by the empty transition sequence at $[\mathsf{C}^{abort} \oplus \mathsf{C}^{commit}]$.

For (3), please consider the collection $\{[\mathsf{C}^{commit}_{pre,t}]\}_{t>0}$. $[\mathsf{C}^{commit}_{pre,t+1}]$ has exactly one non H-hidden, non-input transition, namely $[\mathsf{C}^{commit}_{pre,t+1}] \xrightarrow{\tau}_{a} [\mathsf{C}^{commit}_{pre,t}]$ which is matched by the empty transition sequence at $[!\overline{abort} \oplus !\overline{commit}]$. $[\mathsf{C}^{commit}_{pre,1}]$ has also exactly one non H-hidden, non-input transition:

$$[\mathsf{C}^{commit}_{pre,1}] \xrightarrow{\tau}_{a} (\nu ca)(\overline{a} \,|\, c.\mathsf{C}^{commit} \,|\, a.\mathsf{C}^{abort})] \approx^{H}_{a} [\mathsf{C}^{abort}]_{A}$$

This means, we can combine (1) and (2) with the help of Theorem 37.2 to get (3). □

6.5 Concluding Remarks

This chapter has adapted the 2PCP and its correctness proof from Chapter 3 to the possibility of non-byzantine message failure. This essentially meant adding Recursive Timers to detect and mask message loss. Message duplication played no significant role as a source for failures because the 2PCP is input-affine. The original proof remained largely unchanged except where Recursive Timers introduced new intermediate states that need considerable efforts to be taken care of.

Chapter 7

Persistence and Process Failures

This chapter introduces our final extension: location failure and savepoints.

7.1 Introduction

Message loss is not the only problem for DS. Machines and processes can fail or crash. This is not specific to DS: lacking a notion of partial failure, when a centralised system crashes, the whole computational process comes to an end. But when computation is made of discrete and fairly independent parts, it makes sense to consider the possibility that some of those parts may fail while others don't. What semantic effect would such partial failure have? What mechanisms would be appropriate to deal with partial failure?

This chapter has a first stab at answering such questions in a rigorously formal sense. Unlike with timers and message failure, our treatment of process failures will be superficial, because we do not think to have come anywhere near a good enough understanding of all the issues involved. In a stronger sense than in other parts of this document, our development here has been guided by a desire to provide a minimal amount of technology sufficient for the convenient presentation of the full 2PCP, which can also overcome process failure. In particular, we do not investigate the canonical reduction congruence, although that would be an interesting subject of study.

We model partial failure at the level of sites. For a process to *fail* or *crash* (henceforth we shall use these two terms interchangeably) means that it cannot participate in interactions and that it cannot perform any reductions until it restarts (which it might or might not do). One may think of crashed processes as those that, before restarting, act like the $\mathsf{0}$ process. Failure and restart are allowed at any point in time. Since there is nothing a process can do to influence failure or restart, failures and restarts of sites are completely non-deterministic events. More precisely, we assume that failure cannot occur *during* an interaction, that is, we assume the

all computational steps to be atomic. This gives a fairly accurate approximation to the behaviour of distributed systems under the assumption that no byzantine failure is possible.

One way of coping with the possibility of non-deterministic site failures and restarts, and the one that we are going to formalise, is to allow processes to specify how to restart, if a restart happens, but not if or when. This can be achieved in many ways that often boil down to the existence of *persistent memory* that is (assumed to be) unaffected by failures, together with mechanism that allows restarting processes to read data off that persistent store to find out as what kind of process it should re-emerge. This means splitting computation in two: subject to failure and failure-free. There are more than one way to save process state: operating systems often save entire processes for scheduling purposes at run time. A weaker mechanism would allow 'passive data' to be made persistent, for example indications that a process has passed a certain program point. Data that is made persistent to aid later restart is called a *savepoint*. So in some sense we overcome some of the problems of distribution by even more distribution. This sounds paradoxical unless one notices an asymmetry: the part of the computation that is not subject to failure is passive, it is just data sitting around, not doing very much, unlike the failure-prone part. Activity seems to incur more of a danger of failure than passivity.

Our model of site failure and restart is syntactically straightforward. We start with π_{mlt}, although any calculus with a reasonable notion of location would do just as well. At the level of processes, we introduce a new prefix for making savepoints. As π-calculi do not distinguish state, data and processes, we allow processes to be savepoints.

$$\mathsf{P} \quad ::= \quad \ldots \mid \mathsf{save}(\mathsf{P}).\mathsf{Q}.$$

Each site records the latest savepoint in a superscript, with new savepoints overwriting previous ones. Additionally, we need to represent a crashed process that has not yet restarted. We denote such a process by $[\star]^{\mathsf{P}}_A$. Networks are now given by the following grammar.

$$\mathsf{N} \quad ::= \quad \ldots \mid [\mathsf{P}]^{\mathsf{Q}}_A \mid [\star]^{\mathsf{P}}_A$$

A process $[\mathsf{P}]^{\mathsf{Q}}_A$ should be understood as a site containing a process P with the latest savepoint being Q. Should it crash, it will become $[\star]^{\mathsf{Q}}_A$. A site $[\star]^{\mathsf{Q}}_A$ represents a crashed process that will become $[\mathsf{Q}]^{\mathsf{Q}}_A$ should it ever restart. We do not require Q to have any resemblance to P.

7.2 The Calculus

Just as in the case of π_{mlt}, the calculus is a two-sorted algebra with *processes*, ranged over by $\mathsf{P}, \mathsf{Q}, \mathsf{R}, ...$, given by the following grammar

P	$::=$	$x(\vec{v}).\mathsf{P}$	input
	$\mid$	$\overline{x}\langle\vec{y}\rangle$	output
	$\mid$	$\mathsf{save}(\mathsf{P}).\mathsf{Q}$	savepoint
	$\mid$	$\mathsf{P} \mid \mathsf{Q}$	parallel composition
	$\mid$	$(\nu x)\mathsf{P}$	restriction
	$\mid$	$!x(\vec{v}).\mathsf{P}$	replication
	$\mid$	$\mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q})$	timer
	$\mid$	$\mathsf{0}$	inactive process

Networks, as before ranged over by M, N, are defined by the next grammar.

N	$::=$	$[\mathsf{P}]^{\mathsf{Q}}$	running process
	$\mid$	$[\star]^{\mathsf{P}}_A$	crashed process
	$\mid$	$(\nu x)\mathsf{N}$	restriction
	$\mid$	$\mathsf{N}\mid\mathsf{N}$	parallel composition
	$\mid$	$\mathsf{0}$	inaction

The free and bound names of processes and networks are summarised in Figure 7.1, together with the definition of the structural congruence.

The well-formedness conditions, formalised in π_{mlt} with a predicate $\vdash$ are extended straightforwardly to deal with process failure and savepoints.

Definition 69 We say N is *well-formed*, written $\vdash \mathsf{N}$, if $\vdash \mathsf{N}$ is derivable using the following rules.

- $\vdash \mathsf{0}$ is always derivable.
- $\vdash [\mathsf{P}]^{\mathsf{Q}}_A$ if P as well as Q are local and each free input subject in P and in Q is in A.
- $\vdash [\star]^{\mathsf{P}}_A$ if P is local and each free input subject in P is in A.
- $\vdash \mathsf{M}|\mathsf{N}$ if $\vdash \mathsf{M}$ and $\vdash \mathsf{N}$ and, moreover, $\mathsf{ap}(\mathsf{M}) \cap \mathsf{ap}(\mathsf{N}) = \emptyset$.
- $\vdash (\nu x)\mathsf{N}$ if $\vdash \mathsf{N}$.

where the *access points* $\mathsf{ap}(\mathsf{N})$ of a network N are given by: $\mathsf{ap}([\mathsf{P}]^{\mathsf{Q}}_A) = A$, $\mathsf{ap}([\star]^{\mathsf{P}}_A) = A$, $\mathsf{ap}(\mathsf{N}_1|\mathsf{N}_2) = \mathsf{ap}(\mathsf{N}_1) \cup \mathsf{ap}(\mathsf{N}_2)$ and $\mathsf{ap}((\nu x)\mathsf{N}) = \mathsf{ap}(\mathsf{N}) \backslash \{x\}$. In addition, the notion of locality from Definition 54 in §5.2 is extended to include savepointing by saying that $\mathsf{save}(\mathsf{P}).\mathsf{Q}$ is *local* if both, P and Q are local.

$$
\begin{array}{lcllcl}
\mathsf{fn}(\mathsf{save}(\mathsf{P}).\mathsf{Q}) & = & \mathsf{fn}(\mathsf{save}(\mathsf{P})) \cup \mathsf{fn}(\mathsf{Q}) & \mathsf{bn}(\mathsf{save}(\mathsf{P}).\mathsf{Q}) & = & \mathsf{bn}(\mathsf{save}(\mathsf{P})) \cup \mathsf{bn}(\mathsf{Q}) \\
\mathsf{fn}(\mathsf{timer}^t(\mathsf{P},\mathsf{Q})) & = & \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q}) & \mathsf{bn}(\mathsf{timer}^t(\mathsf{P},\mathsf{Q})) & = & \mathsf{bn}(\mathsf{P}) \cup \mathsf{bn}(\mathsf{Q}) \\
\mathsf{fn}(\overline{x}\langle\vec{y}\rangle) & = & \{x,\vec{y}\} & \mathsf{bn}(\overline{x}\langle\vec{y}\rangle) & = & \emptyset \\
\mathsf{fn}(x(\vec{y}).\mathsf{P}) & = & \{x\} \cup (\mathsf{fn}(\mathsf{P}) \setminus \{\vec{y}\}) & \mathsf{bn}(x(\vec{y}).\mathsf{P}) & = & \mathsf{bn}(\mathsf{P}) \cup \{\vec{y}\} \\
\mathsf{fn}(!x(\vec{v}).\mathsf{P}) & = & \{x\} \cup (\mathsf{fn}(\mathsf{P}) \setminus \{\vec{y}\}) & \mathsf{bn}(!x(\vec{v}).\mathsf{P}) & = & \{x\} \cup (\mathsf{fn}(\mathsf{P}) \setminus \{\vec{y}\}) \\
\mathsf{fn}(\mathsf{P}|\mathsf{Q}) & = & \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q}) & \mathsf{bn}(\mathsf{P}|\mathsf{Q}) & = & \mathsf{bn}(\mathsf{P}) \cup \mathsf{bn}(\mathsf{Q}) \\
\mathsf{fn}((\nu x)\mathsf{P}) & = & \mathsf{fn}(\mathsf{P}) \setminus \{x\} & \mathsf{bn}((\nu x)\mathsf{P}) & = & \mathsf{bn}(\mathsf{P}) \cup \{x\} \\
\mathsf{fn}(0) & = & \emptyset & \mathsf{bn}(0) & = & \emptyset
\end{array}
$$

$$
\begin{array}{lcllcl}
\mathsf{fn}(\overline{x}\langle\vec{y}\rangle) & = & \{x,\vec{y}\} & \mathsf{bn}(\overline{x}\langle\vec{y}\rangle) & = & \emptyset \\
\mathsf{fn}([\mathsf{P}]_A^{\mathsf{Q}}) & = & \mathsf{fn}(\mathsf{P}) \cup A \cup \mathsf{fn}(\mathsf{Q}) & \mathsf{bn}([\mathsf{P}]_A^{\mathsf{Q}}) & = & \mathsf{bn}(\mathsf{P}) \cup \mathsf{bn}(\mathsf{Q}) \\
\mathsf{fn}(\mathsf{M} \mid \mathsf{N}) & = & \mathsf{fn}(\mathsf{M}) \cup \mathsf{fn}(\mathsf{N}) & \mathsf{bn}(\mathsf{M} \mid \mathsf{N}) & = & \mathsf{bn}(\mathsf{M}) \cup \mathsf{bn}(\mathsf{N}) \\
\mathsf{fn}((\nu x)\mathsf{N}) & = & \mathsf{fn}(\mathsf{N}) \setminus \{x\} & \mathsf{bn}((\nu x)\mathsf{N}) & = & \mathsf{bn}(\mathsf{N}) \cup \{x\} \\
\mathsf{fn}(0) & = & \emptyset & \mathsf{bn}(0) & = & \emptyset
\end{array}
$$

$\equiv$ is the least congruence satisfying the following rules.

$$
\begin{array}{ll}
\mathsf{P} \equiv_\alpha \mathsf{Q} \Rightarrow \mathsf{P} \equiv \mathsf{Q} & (\nu x)0 \equiv 0 \\
\mathsf{P}|\mathsf{Q} \equiv \mathsf{Q}|\mathsf{P} & \mathsf{P}|0 \equiv \mathsf{P} \\
\mathsf{P}|(\mathsf{Q}|\mathsf{R}) \equiv (\mathsf{P}|\mathsf{Q})|\mathsf{R} & x \notin \mathsf{fn}(\mathsf{P}) \Rightarrow \mathsf{P}|(\nu x)\mathsf{Q} \equiv (\nu x)(\mathsf{P}|\mathsf{Q}) \\
(\nu x)(\nu y)\mathsf{P} \equiv (\nu y)(\nu x)\mathsf{P} &
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{M} \equiv_\alpha \mathsf{N} \Rightarrow \mathsf{M} \equiv \mathsf{N} & \mathsf{M} \mid \mathsf{N} \equiv \mathsf{N} \mid \mathsf{M} \\
\mathsf{L} \mid (\mathsf{M} \mid \mathsf{N}) \equiv (\mathsf{L} \mid \mathsf{M}) \mid \mathsf{N} & \mathsf{M} \mid 0 \equiv \mathsf{M} \\
x \notin \mathsf{fn}(\mathsf{M}) \Rightarrow \mathsf{M} \mid (\nu x)\mathsf{N} \equiv (\nu x)(\mathsf{M} \mid \mathsf{N}) & (\nu x)(\nu y)\mathsf{M} \equiv (\nu y)(\nu x)\mathsf{M} \\
(\nu x)0 \equiv 0 & [(\nu x)\mathsf{P}]_A \equiv (\nu x)[\mathsf{P}]_{A \cup \{x\}} \\
[0]_\emptyset \equiv 0 & \mathsf{P} \equiv \mathsf{Q} \Rightarrow [\mathsf{P}]_A \equiv [\mathsf{Q}]_A
\end{array}
$$

Figure 7.1: The inductive definition of the free and bound names as well as the structural congruence.

7.3 Semantics

7.3.1 Dynamics and Structural Congruence

The reduction relation is defined in Figures 7.1, 7.2 and 7.3. With the exception of three new rules that we will explain now, the definition is essentially that of Figure 5.1 for π_{mlt}.

The first new rule shows how taking savepoints works.

$$(\textsc{Save}) \quad \frac{}{[\mathsf{P} \mid \mathsf{save}(\mathsf{Q}).\mathsf{R}]^{\mathsf{S}}_A \to [\mathsf{P} \mid \mathsf{R}]^{\mathsf{Q}}_A}$$

In the context of a site, the program $\mathsf{save}(\mathsf{Q}).\mathsf{R}$ does essentially two things: it saves the process Q in permanent storage and it releases the continuation R. One may ask if this introduces synchrony into an otherwise asynchronous calculus. We are not so sure. The very concept is usually defined with respect to input and output, in ways that are not directly applicable here. On the other hand, it seems vital for consistent restarts, to combine savepointing with sequencing. This will become clearer in the next chapter.

Process failure is modelled by the (Stop) rule given next.

$$(\textsc{Stop}) \quad \frac{}{[\mathsf{P}]^{\mathsf{Q}}_A \to [\star]^{\mathsf{Q}}_A}$$

All it does is erase any trace of the active process P. As you can see, the savepoint Q is unaffected by the crash. To restart a process we use the dual (Start) rule, which simply plugs the saved process into the site.

$$(\textsc{Start}) \quad \frac{}{[\star]^{\mathsf{P}}_A \to [\mathsf{P}]^{\mathsf{P}}_A}$$

These rules ensure that processes can always crash and restart. This is a vital assumption for the 2PCP, but alternatives are conceivable.

We could now define all manner of semantic technology, as we have done in previous chapters. But there is no need here, because we do not develop the theory of reduction congruence here. Instead we only introduce those constructs needed in the next chapter.

Definition 70 *Process contexts* are inductively given (up to $\equiv$) by the following grammar, extending that in §4.3.1.

$$C[\cdot] ::= \ldots \mid \mathsf{save}(C[\cdot]).\mathsf{P} \mid \mathsf{save}(\mathsf{P}).C[\cdot]$$

$$\text{(Com)}\quad \frac{}{x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \;\rightarrow\; \mathsf{P}\{\vec{v}/\vec{y}\}}$$

$$\text{(Rep)}\quad \frac{}{!x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \rightarrow !x(\vec{v}).\mathsf{P} \mid \mathsf{P}\{\vec{v}/\vec{y}\}}$$

$$\text{(TimeIn)}\quad \frac{}{\mathsf{timer}^{t+1}(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \mid \overline{x}\langle\vec{y}\rangle \rightarrow \mathsf{P}\{\vec{y}/\vec{v}\}}$$

$$\text{(Par)}\quad \frac{\mathsf{P} \rightarrow \mathsf{P}'}{\mathsf{P} \mid \mathsf{Q} \rightarrow \mathsf{P}' \mid \phi(\mathsf{Q})}$$

$$\text{(Res)}\quad \frac{\mathsf{P} \rightarrow \mathsf{Q}}{(\nu x)\mathsf{P} \rightarrow (\nu x)\mathsf{Q}}$$

$$\text{(Idle)}\quad \frac{}{\mathsf{P} \rightarrow \phi(\mathsf{P})}$$

Figure 7.2: The inductive definition of the reduction relation.

Free Input and Output Names

The functions $\mathsf{fin}(\cdot)$ and $\mathsf{fon}(\cdot)$ are straightforward extensions of their π_{mlt} counterparts with the following new clauses

$$\begin{array}{lcllcl} \mathsf{fin}(\mathsf{save}(\mathsf{P}).\mathsf{Q}) & = & \mathsf{fin}(\mathsf{P}) \cup \mathsf{fin}(\mathsf{Q}) & \mathsf{fon}(\mathsf{save}(\mathsf{P}).\mathsf{Q}) & = & \mathsf{fon}(\mathsf{P}) \cup \mathsf{fon}(\mathsf{Q}) \\ \mathsf{fin}([\mathsf{P}]_A^{\mathsf{Q}}) & = & \mathsf{fin}(\mathsf{P}) \cup \mathsf{fin}(\mathsf{Q}) & \mathsf{fon}([\mathsf{P}]_A^{\mathsf{Q}}) & = & \mathsf{fon}(\mathsf{P}) \cup \mathsf{fon}(\mathsf{Q}) \end{array}$$

For consistency, P's free input and output names in $\mathsf{save}(\mathsf{P}).\mathsf{Q}$ are also counted towards $\mathsf{fin}(\mathsf{save}(\mathsf{P}).\mathsf{Q})$ and $\mathsf{fon}(\mathsf{save}(\mathsf{P}).\mathsf{Q})$, respectively, despite no process in the same site as $\mathsf{save}(\mathsf{P}).\mathsf{Q}$ being able to interact with P. For a finer analysis, one would need to be more careful with what counts as free names, but this is not necessary here.

7.3.2 Transitional Semantics

To obtain labelled semantics for our new calculus we add labels of the form $\mathsf{save}(\mathsf{P})$ (where P is any process) to those of the π-calculi already discussed. *Actions* are then generated by the following grammar.

$$\pi ::= \overline{x}\langle(\nu\vec{y})\vec{z}\rangle \;\mid\; x((\nu\vec{y})\vec{z}) \;\mid\; \mathsf{save}(\mathsf{P}) \;\mid\; \tau$$

The free and bound names of $\mathsf{save}(\mathsf{P})$ are (not surprisingly) given by $\mathsf{fn}(\mathsf{save}(\mathsf{P})) = \mathsf{fn}(\mathsf{P})$ and $\mathsf{bn}(\mathsf{save}(\mathsf{P})) = \mathsf{bn}(\mathsf{P})$.

$$\text{(Save)}\quad \frac{}{[\mathsf{P} \mid \mathsf{save}(\mathsf{Q}).\mathsf{R}]^{\mathsf{S}}_A \to [\mathsf{P} \mid \mathsf{R}]^{\mathsf{Q}}_A}$$

$$\text{(Intra)}\quad \frac{\mathsf{P} \to \mathsf{Q}}{[\mathsf{P}]^{\mathsf{R}}_A \to [\mathsf{Q}]^{\mathsf{R}}_A}$$

$$\text{(Stop)}\quad \frac{}{[\mathsf{P}]^{\mathsf{Q}}_A \to [\star]^{\mathsf{Q}}_A}$$

$$\text{(Start)}\quad \frac{}{[\star]^{\mathsf{P}}_A \to [\mathsf{P}]^{\mathsf{P}}_A}$$

$$\text{(Send)}\quad \frac{x \notin A}{[\overline{x}\langle\vec{y}\rangle \mid \mathsf{P}]^{\mathsf{Q}}_A \to [\phi(\mathsf{P})]^{\mathsf{Q}}_A \mid \overline{x}\langle\vec{y}\rangle}$$

$$\text{(Get)}\quad \frac{x \in A}{[\mathsf{P}]_A \mid \overline{x}\langle\vec{y}\rangle \to [\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle]_A}$$

$$\text{(Loss)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \to \mathsf{0}}$$

$$\text{(Dupl)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \to \overline{x}\langle\vec{y}\rangle \mid \overline{x}\langle\vec{y}\rangle}$$

$$\text{(N-Par)}\quad \frac{\mathsf{M} \to \mathsf{M}'}{\mathsf{M} \mid \mathsf{N} \to \mathsf{M}' \mid \mathsf{N}}$$

$$\text{(N-Res)}\quad \frac{\mathsf{M} \to \mathsf{N}}{(\nu x)\mathsf{M} \to (\nu x)\mathsf{N}}$$

$$\text{(Cong)}\quad \frac{\mathsf{M} \equiv \mathsf{M}' \quad \mathsf{M}' \to \mathsf{N}' \quad \mathsf{N}' \equiv \mathsf{N}}{\mathsf{M} \to \mathsf{N}}$$

Figure 7.3: The inductive definition of the reduction relation (continued).

$$\text{(S-Out)}\quad \frac{}{\mathsf{save(P).Q} \xrightarrow{\mathsf{save(P)}}_a \mathsf{Q}}$$

$$\text{(Save)}\quad \frac{\mathsf{P} \xrightarrow{\mathsf{save(S)}}_a \mathsf{Q}}{[\mathsf{P}]^{\mathsf{R}} \xrightarrow{\tau}_a [\mathsf{Q}]^{\mathsf{S}}}$$

$$\text{(Intra)}\quad \frac{\mathsf{P} \xrightarrow{\tau}_a \mathsf{Q}}{[\mathsf{P}]_A \xrightarrow{\tau}_a [\mathsf{Q}]_A}$$

$$\text{(Stop)}\quad \frac{}{[\mathsf{P}]^{\mathsf{P}}_A \xrightarrow{\tau}_a [\star]^{\mathsf{P}}_A}$$

$$\text{(Start)}\quad \frac{}{[\star]^{\mathsf{P}}_A \xrightarrow{\tau}_a [\mathsf{P}]^{\mathsf{P}}}$$

$$\text{(Send)}\quad \frac{x \notin A}{[\mathsf{P}|\overline{x}\langle\vec{y}\rangle]_A \xrightarrow{\tau}_a [\phi(\mathsf{P})]_A \mid \overline{x}\langle\vec{y}\rangle}$$

$$\text{(Get)}\quad \frac{x \in A}{[\mathsf{P}]_A \mid \overline{x}\langle\vec{y}\rangle \xrightarrow{\tau}_a [\mathsf{P}|\overline{x}\langle\vec{y}\rangle]_A}$$

$$\text{(Loss)}\quad \frac{}{\overline{x}\langle\vec{z}\rangle \xrightarrow{\tau}_a \mathsf{0}}$$

$$\text{(Dupl)}\quad \frac{}{\overline{x}\langle\vec{z}\rangle \xrightarrow{\tau}_a \overline{x}\langle\vec{z}\rangle \mid \overline{x}\langle\vec{z}\rangle}$$

$$\text{(EtherIn)}\quad \frac{}{\mathsf{0} \xrightarrow{x(\vec{y})}_a \overline{x}\langle\vec{y}\rangle}$$

$$\text{(EtherOut)}\quad \frac{}{\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{0}}$$

$$\text{(N-Par)}\quad \frac{\mathsf{M} \xrightarrow{l}_a \mathsf{M}' \quad \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{N}) = \emptyset \quad l = \overline{x}\langle(\nu\vec{y})\vec{z}\rangle \Rightarrow x \notin \mathsf{ap}(\mathsf{N})}{\mathsf{M}|\mathsf{N} \xrightarrow{l}_a \mathsf{M}'|\mathsf{N}}$$

$$\text{(N-Res)}\quad \frac{\mathsf{M} \xrightarrow{l}_a \mathsf{N} \quad x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)}{(\nu x)\mathsf{M} \xrightarrow{l}_a (\nu x)\mathsf{N}}$$

$$\text{(N-Open)}\quad \frac{\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{N} \quad v \neq x, v \in \{\vec{z}\} \setminus \{\vec{y}\}}{(\nu v)\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y},v)\vec{z}\rangle}_a \mathsf{N}}$$

$$\text{(N-Cong)}\quad \frac{\mathsf{M} \equiv \mathsf{M}' \quad \mathsf{M}' \xrightarrow{l}_a \mathsf{N}' \quad \mathsf{N} \equiv \mathsf{N}'}{\mathsf{M} \xrightarrow{l}_a \mathsf{N}}$$

Figure 7.4: An asynchronous transition system for the timed, asynchronous π-calculus with locations, message loss and message duplication. Apart from (S-Out), transitions for processes are omitted and can be found in Figure 4.2.

The transition relation is generated inductively by the rules in Figure 7.4 together with those of π_t, cf. Figure 4.3. The rules are all familiar from π_{mlt} with the exception of (S-Out) and (Save), which should be clear from the the explanations above. The (S-Out) rule introduces a form of process passing. It is weaker than process passing in higher order π-calculi [100] because no process can interact with an emission of $\mathsf{save}(\mathsf{P})$. Interaction with such an emission is exclusive to the persistence mechanism integrated into sites.

The next definition – repeated for convenience – is formally identical with Definition 66 but works with respect to a different labelled transition system.

Definition 71 A binary symmetric relation $\mathcal{R}$ on networks is a *strong asynchronous bisimulation* if $(\mathsf{M},\mathsf{N}) \in \mathcal{R}$ and $\mathsf{M} \xrightarrow{l}_a \mathsf{M}'$ implies that some transition $\mathsf{N} \xrightarrow{l}_a \mathsf{N}'$ exists with $(\mathsf{M}',\mathsf{N}') \in \mathcal{R}$. *Asynchronous bisimulations* are obtained analogously.

7.4 Concluding Remarks

This chapter added process failure and savepoints to π_{mlt}, not to study the properties of the resulting calculus in any depth, but rather as a stepping stone towards the full 2PCP. Persistence and savepointing have received much attention from the distributed systems community [33, 47], but to the best of our knowledge process theoretic accounts are lacking. This leaves much work to be done, in particular, a satisfactory theory of the induced reduction congruence. We conjecture that this will be entirely along the lines of what we did for π_{mlt} and hence straightforward. The barbs used for the asynchronous π-calculus, π_t and π_{mlt} are most likely also the right ones for our calculus. An important problem that has to be investigated more thoroughly is the appropriateness of this chapter's extensions. We simply have not got enough experience with the problem domain to confidently pass a judgement. Being able to compare with alternative approaches would be good for a start, but what could those look like in π-calculi? Since names are important, it might be possible to distinguish persistent from volatile names, although details far from clear. Alternatively, we might give up on distinguishing an active but volatile process from a passive, yet persistent process within a site $[\mathsf{P}]^{\mathsf{Q}}_A$. Instead we could distinguish volatile sites $[\mathsf{P}]_A$, subject to process failure, from persistent sites $[[\mathsf{P}]]_A$ which are not. The ability to take savepoints by way of dedicated syntax would no longer be necessary and superseded by appropriate interaction with persistent sites. This would be a more lightweight addition than $\mathsf{save}(\mathsf{P}).\mathsf{Q}$. It would also appear to lead to more realistic models. A variant of this proposal is to use sites $[\mathsf{P} \mid \underline{\mathsf{Q}}]_A$ instead of $[\mathsf{P}]^{\mathsf{Q}}_A$. The idea here is that both P and Q are active and can interact, but the underlined Q is persistent while P is subject to process failure (i.e. we have transitions like $[\mathsf{P} \mid \underline{\mathsf{Q}}]_A \to [\star \mid \underline{\mathsf{Q}}]_A$). This model reflects many contemporary distributed

systems where each physical node has its own persistent storage (the hard-disk or tape-drive). Clearly, the last proposal is a generalisation of distinguishing persistent from volatile sites. It seems likely that primitives like $\mathsf{save}(\mathsf{P}).\mathsf{Q}$ can be encoded into either of the alternatives last sketched.

Regardless of the details, persistent processes will have a peculiar structure which could be described by interesting typing systems. Similarly, and we will discuss this in the next chapter, programs that can guarantee consistent behaviour in the face of process failure must be rather constrained. Suitable constraints should be pinned down by types, too.

Chapter 8

The Two-Phase Commit Protocol

In this chapter, the full 2PCP with message and process failure is presented. In addition, the correctness of the protocol with one participant is verified.

8.1 Introduction

In previous chapters we presented the 2PCP under restricted failure assumptions. Now that we have process failures at our disposal, we can augment the protocol to obtain the full 2PCP of [19, 47]. In comparison with the upgrade from no failures to message failures, the present change requires no modification apart from the occasional judiciously chosen savepoint. The proof will also retain its Divide-And-Conquer structure, but with intermediate equations being significantly more coarse grained. It seems that while process failure requires fewer algorithmic changes than message failure, its effect on proofs is greater. As in Chapter 6, the correctness proof will only tackle the special case of one participant, to keep the presentation manageable.

8.2 The Full 2PCP

Once again, the protocol has n participants and one coordinator, all hosted by dedicated locations. Internal communication happens via the private names $\vec{vote}$, $\vec{dec}$, $\vec{e}$. All processes carry an index, denoting the time units left until time-out of their timers. In addition, each process has a current savepoint that it will restart as, after it re-emerges from a process failure.

$$\mathsf{2PCP}_{t,t'} = (\nu \vec{vote})(\nu \vec{dec})(\nu \vec{e})([\mathsf{C}_t]^{\mathsf{C}^{abort}}_{\{\vec{vote},\vec{e}\}} \mid [\mathsf{P}_{1,t'}]^{\mathsf{P}^{abort}_{1,t'}}_{\{dec_1\}} \mid ... \mid [\mathsf{P}_{n,t'}]^{\mathsf{P}^{abort}_{n,t'}}_{\{dec_n\}})$$

The new challenge is to keep these savepoints up to date to avoid inconsistency. As an example of why that needs some care, consider a potential implementation of a participant, just before it decides how to vote: $[\mathsf{P}_i^a \oplus \mathsf{P}_i^c]_A^{\mathsf{P}_i^a}$. It has the aborting process as a savepoint because it seems reasonable to make a crashed process abort: process failure indicates a serious fault and commitment should only be agreed on if all participants have executed with without encountering problems. But with processes crashing non-deterministically, computations like the following are possible.

$$\begin{array}{lll} [\mathsf{P}_i^a \oplus \mathsf{P}_i^c]_A^{\mathsf{P}_i^a} & \to & [\mathsf{P}_i^c]_A^{\mathsf{P}_i^a} \\ & \xrightarrow{\overline{vote_i}\mathsf{left}}_a & [\ldots]_A^{\mathsf{P}_i^a} \\ & \to & [\star]_A^{\mathsf{P}_i^a} \\ & \to & [\mathsf{P}_i^a]_A^{\mathsf{P}_i^a} \\ & \xrightarrow{\overline{vote_i}\mathsf{right}}_a & [\ldots]_A^{\mathsf{P}_i^a} \end{array}$$

This means we can have processes voting twice and inconsistently. If only the first vote is received by the coordinator and all other participants voted for commitment, the atomicity of the protocol would be broken.

Fortunately, we can use savepointing to update the savepoints in ways that prevent inconsistency. Timing is important for this. Savepoints must be taken immediately after a process has changed its state in ways that have observable effects, but before this state change has been communicated. Relevant sources of state change are internal choices, time-outs or branching inputs. As it is vital to take savepoints immediately, uses of $\mathsf{save}(\cdot)$ will be of the form

- $(\mathsf{save}(\mathsf{P}).\mathsf{P}) \oplus (\mathsf{save}(\mathsf{Q}).\mathsf{Q})$,
- $x[(\vec{v}).\mathsf{save}(\mathsf{P}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{save}(\mathsf{Q}).\mathsf{Q}]$,
- $\mathsf{timer}^t(x(v).\mathsf{save}(\mathsf{P}).\mathsf{P}, \mathsf{save}(\mathsf{Q}).\mathsf{Q})$ or
- $\mathsf{timer}^t(x[(\vec{v}).\mathsf{save}(\mathsf{P}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{save}(\mathsf{Q}).\mathsf{Q}], \mathsf{save}(\mathsf{R}).\mathsf{R})$.

Savepoints that have no semantic effect can be omitted, For example $\mathsf{save}(\mathsf{Q})$ in $[(\mathsf{save}(\mathsf{P}).\mathsf{P}) \oplus (\mathsf{save}(\mathsf{Q}).\mathsf{Q})]_A^{\mathsf{Q}}$ is without semantic effect. Hence we would normally just write $[(\mathsf{save}(\mathsf{P}).\mathsf{P}) \oplus \mathsf{Q}]_A^{\mathsf{Q}}$, etc.

Participants are obtained as just described from those in §6.2 by adding savepoints to the two sources of uncertainty: the internal choice that decides how to vote and the decision message from the coordinator (in case the vote was towards committing). To understand the algorithm, it should be kept in mind that the

initial savepoint of the i^{th} participant is $\mathsf{P}^{abort}_{i,t'}$.

$$\begin{array}{rcl}
\mathsf{P}_{i,t} & = & \mathsf{P}^{abort}_{i} \oplus \mathsf{P}^{precommit}_{i,t} \\
\mathsf{P}^{abort}_{i} & = & \overline{vote_i}\mathsf{right} \mid !\overline{abort_i} \\
\mathsf{P}^{precommit}_{i,t} & = & \mathsf{save}(\mathsf{P}^{commit}_{i,t}).\mathsf{P}^{commit}_{i,t} \\
\mathsf{P}^{commit}_{i,t} & = & \overline{vote_i}\mathsf{left} \mid \mathsf{P}^{wait}_{i,t} \\
\mathsf{P}^{wait}_{i,t} & = & \mathsf{timer}^t(dec_i[\mathsf{P}^{precommit}_{i}, \mathsf{P}^{preabort}_{i}], \overline{e_i} \mid \mathsf{P}^{wait}_{i,t}) \\
\mathsf{P}^{precommit}_{i} & = & \mathsf{save}(!\overline{commit_i}).!\overline{commit_i} \\
\mathsf{P}^{preabort}_{i} & = & \mathsf{save}(!\overline{abort_i}).!\overline{abort_i}
\end{array}$$

The coordinator is derived in the same way. It has two semantically relevant sources of uncertainty that need savepointing. The first is its choice about whether to abort immediately or not. The second is whether all participants vote for commitment *in time*. Of course the coordinator features more branching inputs and timers that on their own would seem prime candidates for uncertainty in need of savepointing. But in the context they find themselves in here, none of them removes uncertainty that would have effects observable on the outside. Hence there is not need for savepointing. Of course more savepoints might help to speed up recovery from process failure, for example by reducing the number of interactions that have to be repeated, but our semantics does not allow to observe network efficiency, so this is not relevant there. With a more fine-grained semantics, for example by having time (partially) synchronised across networks, as alluded to in §5.9, this would change and different savepoints would have to be taken.

$$\begin{array}{rcl}
\mathsf{C}_{t} & = & \mathsf{C}^{abort} \oplus \mathsf{C}^{commit}_{pre,t} \\
\mathsf{C}^{abort} & = & \Pi^{n}_{i=1}\mathsf{C}^{abort}_{i} \\
\mathsf{C}^{abort}_{i} & = & \overline{dec_i}\mathsf{right} \mid !e_i.\overline{dec_i}\mathsf{right}
\end{array}$$

$$\begin{array}{rcl}
\mathsf{C}^{commit}_{pre,t} & = & (\nu \vec{c}a)(\mathsf{C}^{wait}_{t} \mid \mathsf{C}^{and} \mid \mathsf{C}^{or}) \\
\mathsf{C}^{wait}_{t} & = & \Pi^{n}_{i=1}\mathsf{C}^{wait}_{i,t} \\
\mathsf{C}^{wait}_{i,t} & = & \mathsf{timer}^t(vote_i[\overline{c_i}, \overline{a}], \overline{a}) \\
\mathsf{C}^{and} & = & c_1.c_2...c_n.\mathsf{C}^{precommit}_{final} \\
\mathsf{C}^{precommit}_{final} & = & \mathsf{save}(\mathsf{C}^{commit}_{final}).\mathsf{C}^{commit}_{final} \\
\mathsf{C}^{commit}_{final} & = & \Pi^{n}_{i=1}\mathsf{C}^{commit}_{i} \\
\mathsf{C}^{commit}_{i} & = & \overline{dec_i}\mathsf{left} \mid !e_i.\overline{dec_i}\mathsf{left} \\
\mathsf{C}^{or} & = & a.\mathsf{C}^{abort}
\end{array}$$

As before we will often omit the access point of sites.

Remarks on the Underlying Calculus

As in previous chapters, the calculus used here to express the 2PCP is not that of the last chapter, but a variant with binary branching, replicated outputs and definition of processes by recursive equations. As before, the reason for this choice is economy of presentation. All the caveats of §6.2 apply here too.

8.3 Correctness of the 2PCP With One Participant

The meaning of correctness for the 2PCP is hardly changed from Chapter 6, except that we need to take savepoints into account. Of course we cannot simply say $\mathsf{2PCP} \approx_a [\mathsf{Abort} \oplus \mathsf{Commit}]_\emptyset^{\mathsf{Abort}\oplus\mathsf{Commit}}$ as that would not be atomic:

$$\begin{array}{lll} [\mathsf{Abort} \oplus \mathsf{Commit}]_\emptyset^{\mathsf{Abort}\oplus\mathsf{Commit}} & \rightarrow & [\mathsf{Abort}]_\emptyset^{\mathsf{Abort}\oplus\mathsf{Commit}} \\ & \xrightarrow{\overline{abort_i}}_a & [\mathsf{Abort}]_\emptyset^{\mathsf{Abort}\oplus\mathsf{Commit}} \\ & \rightarrow & [\star]_\emptyset^{\mathsf{Abort}\oplus\mathsf{Commit}} \\ & \rightarrow & [\mathsf{Abort} \oplus \mathsf{Commit}]_\emptyset^{\mathsf{Abort}\oplus\mathsf{Commit}} \\ & \xrightarrow{\overline{commit_i}}_a & \ldots \end{array}$$

The problem arises because of not taking a savepoint after removal of uncertainty by internal choice. This is easy to remedy.

DEFINITION 72 $\mathsf{preAbort} = \mathsf{save}(\mathsf{Abort}).\mathsf{Abort}$, $\mathsf{preCommit} = \mathsf{save}(\mathsf{Commit}).\mathsf{Commit}$.

THEOREM 38 *Let* $t, t' > 0$, *then* $\mathsf{2PCP}_{t,t'} \approx_a [\mathsf{preAbort} \oplus \mathsf{preCommit}]_\emptyset^{\mathsf{preAbort}\oplus\mathsf{preCommit}}$.

Incidentally, this theorem shows that we also have to add savepoints to the savepoint, i.e. we ought to have $[\mathsf{preAbort}\oplus\mathsf{preCommit}]_\emptyset^{\mathsf{preAbort}\oplus\mathsf{preCommit}}$, not just $[\mathsf{preAbort}\oplus\mathsf{preCommit}]_\emptyset^{\mathsf{Abort}\oplus\mathsf{Commit}}$, because savepoints may in general also contain sources of uncertainty which can have external effects if the savepoints become active processes. In the transformation of the 2PCP algorithm described in the previous section, this was not relevant because all used savepoints were devoid of uncertainty.

Theorem 38 does not exhaust what can be considered the 2PCP's correctness, so we establish additional equalities, cf. Theorem 34.

THEOREM 39 *For all* $n, t, t' > 0$.

1. $(\nu v\vec{o}te)(\nu d\vec{e}c)(\nu\vec{e})([\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}_{pre,t}^{commit}} \mid [\mathsf{P}_{1,t}^c]^{\mathsf{P}_{1,t}^c} \mid \ldots \mid [\mathsf{P}_{n,t}^c]^{\mathsf{P}_{n,t}^c}) \approx_a [\mathsf{preAbort}\oplus\mathsf{preCommit}]_\emptyset^{\mathsf{preAbort}\oplus\mathsf{preCommit}}$.
2. *If* $\mathsf{P}_i' \in \{\mathsf{P}_{i,t}^a, \mathsf{P}_{i,t}^c\}$, *then* $(\nu v\vec{o}te)(\nu d\vec{e}c)(\nu\vec{e})([\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}_1']^{\mathsf{P}_1'} \mid \ldots \mid [\mathsf{P}_n']^{\mathsf{P}_n'}) \approx_a [\mathsf{Abort}]_\emptyset^{\mathsf{Abort}}$.

3. *If* $\mathsf{P}'_i \in \{\mathsf{P}^a_{i,t}, \mathsf{P}^c_{i,t}\}$ *and for at least one* $i_0 \in \{1, ..., n\}$ $\mathsf{P}'_{i_0} = \mathsf{P}^a_{i_0,t}$, *then*

$$(\nu \vec{vote})(\nu \vec{dec})(\nu \vec{e})([\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}'_1]^{\mathsf{P}'_1} \mid ... \mid [\mathsf{P}'_n]^{\mathsf{P}'_n}) \approx_a [\mathsf{Abort}]^{\mathsf{Abort}}_{\emptyset}.$$

The caveats about divergence apply here too, in particular since process failure introduces another source of divergence.

8.4 Correctness of the 2PCP with One Participant

The previous correctness proof of the 2PCP in π_{mlt} was restricted to one participant to keep notation manageable. Since process failure makes things decidedly worse in this department, we stick with just one participant.

Some Useful Definitions and Lemmas

The definition of S-bisimulation and related notions remain formally unchanged from Chapter 6, except that the underlying transition system has changed.

DEFINITION 73 Let S be a set of names. We call actions l with $\mathsf{fn}(l) \cap S \neq \emptyset$ *S-hidden*. A network is *static* if all of its or its descendants outputs are free outputs.

A binary relation $\mathcal{R}$ on networks is an *asynchronous S-bisimulation* if $(\mathsf{P}, \mathsf{Q}) \in \mathcal{R}$ implies that P as well as Q are static and

- whenever $\mathsf{P} \xrightarrow{l}_a \mathsf{P}'$ and $\mathsf{fn}(l) \cap S = \emptyset$, then $\mathsf{Q} \xrightarrow{\hat{l}}\!\!\!\!\rightarrow_a \mathsf{Q}'$ for some Q' such that $(\mathsf{P}', \mathsf{Q}') \in \mathcal{R}$,
- and vice versa.

By $\approx_a^S$ we denote the largest asynchronous S-bisimulation. The definitions of *strong asynchronous S-bisimulation* and $\sim_a^S$ are similar.

The next two theorems are adaptations of Theorems 14 and 15.

THEOREM 40 $\sim_a^S \subseteq \approx_a^S$.

PROOF: Straightforward. □

THEOREM 41 *Let* $\mathcal{R}^S$ *be one of* $\approx_a^S$ *and* $\sim_a^S$.

1. $\mathsf{M}\, \mathcal{R}^S\, \mathsf{N}$ *implies* $(\nu x)\mathsf{M}\, \mathcal{R}^{S\setminus\{x\}}\, (\nu x)\mathsf{N}$ *and* $\mathcal{R}^{\emptyset} \subseteq \mathcal{R}$.
2. *If* $S \subseteq T$, *then* $\mathcal{R}^S \subseteq \mathcal{R}^T$.
3. *If* $\mathsf{M}\, \mathcal{R}^S\, \mathsf{N}$ *and* $\mathsf{fn}(\mathsf{R}) \cap S = \emptyset$ *then* $\mathsf{M} \mid \mathsf{L}\, \mathcal{R}^S\, \mathsf{N} \mid \mathsf{L}$.

PROOF: Straightforward. □

The following lemma states two simple but very useful facts.

LEMMA 60 *1.* $[\star]^{\mathsf{P}}_A \approx_a [\mathsf{P}]^{\mathsf{P}}_A$.

2. Assume that $x \notin A$ *and* P *contains no active timers. Let* $\Box$ *be one of* $\overline{x}\langle\vec{y}\rangle$, $\overline{x}\mathsf{left}\langle\vec{y}\rangle$, $\overline{x}\mathsf{right}\langle\vec{y}\rangle$, *then* $[\Box \mid \mathsf{P}]^{\mathsf{Q}}_A \approx_a \Box \mid [\mathsf{P}]^{\mathsf{Q}}_A$

PROOF: Straightforward □

The next lemma shows how certain networks can be ignored when they have only H-hidden names.

LEMMA 61 *Let* H *be a set of names.*

1. *If* $S \subseteq H$ *and* $\mathsf{M} \xrightarrow{l}_a \mathsf{N}$ *is not* H*-hidden, then* $\mathsf{fn}(\mathsf{M}) \cap S = \emptyset$ *implies* $\mathsf{fn}(\mathsf{N}) \cap S = \emptyset$.
2. *Let* $\mathsf{fn}(\mathsf{M}) \subseteq H$ *and* $\mathsf{fn}(\mathsf{M}) \cap \mathsf{fn}(\mathsf{N}) = \emptyset$. *Then* $\mathsf{M} \mid \mathsf{N} \approx^H_a \mathsf{N}$.
3. *If* $\mathsf{fn}(\mathsf{M}) \subseteq H$, $x \in \mathsf{fn}(\mathsf{M}) \Rightarrow x \notin \mathsf{fin}(\mathsf{P}) \cup \mathsf{fin}(\mathsf{Q})$, *and for all* $i \in I$*:* $a_i \notin \mathsf{fin}(\mathsf{P})$, *then*
$$\mathsf{M} \mid [\Pi_{i\in I}\overline{a_i}\langle\vec{b_i}\rangle \mid \mathsf{P}]^{\mathsf{Q}}_A \approx^H_a [\mathsf{P}]^{\mathsf{Q}}_A.$$

PROOF: A straightforward induction on the derivation of $\mathsf{M} \xrightarrow{l}_a \mathsf{N}$ shows (1). The key insight making the induction work is that l's being H-hidden precludes the import of names in S.

For (2) we define $\mathcal{R}$ up to $\equiv$ by

$$\mathsf{M} \mid \mathsf{N}\ \mathcal{R}\ \mathsf{N} \qquad \text{whenever } \mathsf{fn}(\mathsf{M}) \subseteq H \text{ and } \mathsf{fn}(\mathsf{M}) \cap \mathsf{fn}(\mathsf{N}) = \emptyset.$$

Since by construction, M cannot interact with N and all of M's transitions are H hidden, $\mathsf{M} \mid \mathsf{N}$ has at most the following relevant transitions.

- $\mathsf{M} \mid \mathsf{N} \xrightarrow{\tau}_a \mathsf{M}' \mid \mathsf{N}$ because $\mathsf{M} \xrightarrow{\tau}_a \mathsf{M}$. It is matched by the empty transition sequence at N.
- $\mathsf{M} \mid \mathsf{N} \xrightarrow{l}_a \mathsf{M} \mid \mathsf{N}'$ because $\mathsf{N} \xrightarrow{l}_a \mathsf{N}'$. By (1), $\mathsf{fn}(\mathsf{M}) \cap \mathsf{fn}(\mathsf{N}') = \emptyset$, so $\mathsf{N} \xrightarrow{l}_a \mathsf{N}'$ is a match.

Conversely, every non H-hidden transition $\mathsf{N} \xrightarrow{l}_a \mathsf{N}'$ is matched by $\mathsf{M} \mid \mathsf{N} \xrightarrow{l}_a \mathsf{M} \mid \mathsf{N}'$. Since clearly $\mathsf{ap}(\mathsf{M} \mid \mathsf{N}) \setminus H = \mathsf{ap}(\mathsf{N}) \setminus H$ whenever $\mathsf{fn}(\mathsf{M}) \subseteq H$, $\mathcal{R}$ is an asynchronous H-bisimulation.

The proof of (3) utilises $\mathcal{R}$, defined up to $\equiv$ by the following rules.

$$\left.\begin{array}{l} \mathsf{M} \mid [\Pi_{i\in I}\overline{a_i}\langle\vec{b_i}\rangle \mid \mathsf{P}]^{\mathsf{Q}}_A \mid \Pi_{j\in J}\overline{c_j}\langle\vec{d_j}\rangle \\ \\ [\star]^{\mathsf{Q}}_A \mid \Pi_{j\in J}\overline{c_j}\langle\vec{d_j}\rangle \end{array}\right\} \mathcal{R} \left\{\begin{array}{l} [\mathsf{P}]^{\mathsf{Q}}_A \mid \Pi_{j\in J}\overline{c_j}\langle\vec{d_j}\rangle \\ \\ [\star]^{\mathsf{Q}}_A \mid \Pi_{j\in J}\overline{c_j}\langle\vec{d_j}\rangle \end{array}\right.$$

A straightforward check of all possible transitions for the networks related by $\mathcal{R}$ completes the proof. □

We can often ignore inputs and outputs when they cannot possible find someone to interact with. The next lemma shows how.

LEMMA 62 *Let $C[\cdot]$ be a process context not binding x Assume $\Box$ ranges over $\{\overline{x}\langle\vec{y}\rangle, \overline{x}\mathsf{left}\langle\vec{y}\rangle, \overline{x}\mathsf{right}\langle\vec{y}\rangle\}$ and $\triangle$ over $\{x(\vec{v}).\mathsf{P}, x[(\vec{v}).\mathsf{P}\ \&\ (\vec{w}).\mathsf{Q}]\}$.*

1. *Let $x \in H$, but $x \notin \mathsf{fin}(\mathsf{M}) \cup \mathsf{fin}(C[\cdot])$. Then*
   - $[C[\Box]]^{\mathsf{P}}_A \mid \mathsf{M} \sim_a [C[\mathsf{0}]]^{\mathsf{P}}_A \mid \mathsf{M}$.
   - $[\mathsf{P}]^{C[\Box]}_A \mid \mathsf{M} \sim_a [\mathsf{P}]^{C[\mathsf{0}]}_A \mid \mathsf{M}$.
2. *Let $x \in H$, but $x \notin \mathsf{fon}(C[\cdot])$. Then*
   - $[C[\triangle]]^{\mathsf{R}}_A \mid \mathsf{M} \sim_a [C[\mathsf{0}]]^{\mathsf{R}}_A \mid \mathsf{M}$.
   - $[\mathsf{R}]^{C[\triangle]}_A \mid \mathsf{M} \sim_a [\mathsf{R}]^{C[\mathsf{0}]}_A \mid \mathsf{M}$.
3. *Let $x \in H$, but $x \notin \mathsf{fin}(\mathsf{M}) \cup \mathsf{fin}(C[\cdot])$. Then*
   - $[\mathsf{P} \mid (\nu x)C[\Box]]^{\mathsf{Q}}_A \sim_a [\mathsf{P} \mid C[\mathsf{0}]]^{\mathsf{Q}}_A$.
   - $[\mathsf{P}]^{\mathsf{Q} \mid (\nu x)C[\Box]}_A \sim_a [\mathsf{P}]^{\mathsf{Q} \mid C[\mathsf{0}]}_A$.
4. *Let $C[\cdot]$ be a process context not binding x If $x \notin \mathsf{fon}(C[\cdot])$, then*
   - $[\mathsf{P} \mid (\nu x)C[\triangle]]^{\mathsf{R}}_A \sim_a [\mathsf{P} \mid C[\mathsf{0}]]^{\mathsf{R}}_A$.
   - $[\mathsf{P}]^{\mathsf{R} \mid (\nu x)C[\triangle]}_A \sim_a [\mathsf{P}]^{\mathsf{R} \mid C[\mathsf{0}]}_A$.

PROOF: The verification of this lemma is straightforward because by construction neither $\Box$ nor $\triangle$ can find anybody to interact with. We omit the straightforward but tedious details. □

Of course cuts can be eliminated under $\approx_a$, just as in all other π-calculi.

LEMMA 63 *Assume x is fresh, then*

$$[(\nu x)(\overline{x}\langle\vec{y}\rangle \mid x(\vec{v}).\mathsf{P})]^{\mathsf{P}\{\vec{y}/\vec{v}\}}_A \approx_a [\mathsf{P}\{\vec{y}/\vec{v}\}]^{\mathsf{P}\{\vec{y}/\vec{v}\}}_A.$$

PROOF: We omit the proof as it is straightforward. □

The next lemma shows how the Recursive Timers work in the presence of process failure: essentially unchanged. The proof is very tedious but played through for once.

LEMMA 64 *Let $A, B \subseteq H$ and x, e fresh. Assume that $0 < t \leq t'$.*

1. *Let* $\mathsf{P}_{t,t'} = \mathsf{timer}^t(x(\vec{v}).\mathsf{save}(\mathsf{Q}).\mathsf{Q}, \overline{e} \mid \mathsf{P}_{t',t'})$. *Then*

$$[\overline{x}\langle\vec{y}\rangle \mid !e.\overline{x}\langle\vec{y}\rangle]_A^{\overline{x}\langle\vec{y}\rangle \mid !e.\overline{x}\langle\vec{y}\rangle} \mid [\mathsf{P}_{t,t'}]_B^{\mathsf{P}_{t',t'}} \approx_a^H [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}}$$

2. *Let* $\mathsf{P}_{t,t'} = \mathsf{timer}^t(x[(\vec{v}).\mathsf{save}(\mathsf{Q}).\mathsf{Q} \;\&\; (\vec{w}).\mathsf{save}(\mathsf{R}).\mathsf{R}], \overline{e} \mid \mathsf{P}_{t',t'})$. *Then*

$$[\overline{x}\mathsf{left}\langle\vec{y}\rangle \mid !e.\overline{x}\mathsf{left}\langle\vec{y}\rangle]_A^{\overline{x}\mathsf{left}\langle\vec{y}\rangle \mid !e.\overline{x}\mathsf{left}\langle\vec{y}\rangle} \mid [\mathsf{P}_{t,t'}]_B^{\mathsf{P}_{t',t'}} \approx_a^H [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}}$$

3. *Let* $\mathsf{P}_{t,t'} = \mathsf{timer}^t(x[(\vec{v}).\mathsf{save}(\mathsf{Q}).\mathsf{Q} \;\&\; (\vec{w}).\mathsf{save}(\mathsf{R}).\mathsf{R}], \overline{e} \mid \mathsf{P}_{t',t'})$. *Then*

$$[\overline{x}\mathsf{right}\langle\vec{y}\rangle \mid !e.\overline{x}\mathsf{right}\langle\vec{y}\rangle]_A^{\overline{x}\mathsf{right}\langle\vec{y}\rangle \mid !e.\overline{x}\mathsf{right}\langle\vec{y}\rangle} \mid [\mathsf{P}_{t,t'}]_B^{\mathsf{P}_{t',t'}} \approx_a^H [\mathsf{R}\{\vec{y}/\vec{v}\}]_B^{\mathsf{R}\{\vec{y}/\vec{v}\}}$$

PROOF: Let $\Box_i$ be one of $\overline{a_i}\langle\vec{b}\rangle$, $\overline{a_i}\mathsf{left}\langle\vec{b}\rangle$, $\overline{a_i}\mathsf{right}\langle\vec{b}\rangle$, such that $\mathsf{fn}(\Box_i)$, defined in the obvious way, is disjoint from H. We start with a couple of abbreviations.

$$\begin{aligned}
\mathsf{L}_{k,m} &= [\Pi_{i=1}^k \overline{x}\langle\vec{y}\rangle \mid \Pi_{i=1}^m \overline{e} \mid !e.\overline{x}\langle\vec{y}\rangle]_A^{\overline{x}\langle\vec{y}\rangle \mid !e.\overline{x}\langle\vec{y}\rangle} \\
\mathsf{M}_{n,p,q} &= \Pi_{i=1}^n \overline{e} \mid \Pi_{i=1}^p \overline{x}\langle\vec{y}\rangle \mid \Pi_{i=1}^q \Box_i \\
\mathsf{N}_{r,s,t} &= [\Pi_{i=1}^r \overline{e} \mid \Pi_{i=1}^s \overline{x}\langle\vec{y}\rangle \mid \mathsf{P}_{t,t'}]_B^{\mathsf{P}_{t',t'}} \\
\mathsf{N}'_{r,s} &= [\Pi_{i=1}^r \overline{e} \mid \Pi_{i=1}^s \overline{x}\langle\vec{y}\rangle \mid \mathsf{save}(\mathsf{Q}\{\vec{y}/\vec{v}\}).\mathsf{Q}\{\vec{y}/\vec{v}\}]_B^{\mathsf{P}_{t',t'}}
\end{aligned}$$

Define $\mathcal{R}$ up to $\equiv$ by

$$\left.\begin{array}{l}
\mathsf{L}_{k,m} \mid \mathsf{M}_{n,p,q} \mid \mathsf{N}_{r,s,t} \\
[\star]_A^{\overline{x}\langle\vec{y}\rangle \mid !e.\overline{x}\langle\vec{y}\rangle} \mid \mathsf{M}_{n,p,q} \mid \mathsf{N}_{r,s,t} \\
\mathsf{L}_{k,m} \mid \mathsf{M}_{n,p,q} \mid [\star]_B^{\mathsf{P}_{t',t'}} \\
[\star]_A^{\overline{x}\langle\vec{y}\rangle \mid !e.\overline{x}\langle\vec{y}\rangle} \mid \mathsf{M}_{n,p,q} \mid [\star]_B^{\mathsf{P}_{t',t'}} \\
\mathsf{L}_{k,m} \mid \mathsf{M}_{n,p,q} \mid \mathsf{N}'_{r,s} \\
[\star]_A^{\overline{x}\langle\vec{y}\rangle \mid !e.\overline{x}\langle\vec{y}\rangle} \mid \mathsf{M}_{n,p,q} \mid \mathsf{N}'_{r,s}
\end{array}\right\} \mathcal{R} \left\{\begin{array}{l}
[\mathsf{P}\{\vec{y}/\vec{v}\}]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \mid \Pi_{i=1}^q \Box_i \\
[\star]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \mid \Pi_{i=1}^q \Box_i
\end{array}\right.$$

We will now show $\mathcal{R} \cup \approx_a^H$ to be an asynchronous H-bisimulation. First note that six networks on the left of the definition of $\mathcal{R}$ can all reduce to either network on the right. This means that every transition of the latter is easily matched by the former. On the other hand, $\mathsf{K} \stackrel{def}{=} \mathsf{L}_{k,m} \mid \mathsf{M}_{n,p,q} \mid \mathsf{N}_{r,s,t}$ has the following transitions.

- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{K}$, an idle step of the left site.
- $\mathsf{K} \xrightarrow{\tau}_a [\star]_A^{\overline{x}\langle\vec{y}\rangle \mid !e.\overline{x}\langle\vec{y}\rangle} \mid \mathsf{M}_{n,p,q} \mid \mathsf{N}_{r,s,t}$, a crash of the left site.

- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, [\star]_B^{\mathsf{P}_{t',t'}}$, a crash of the right site.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k+1,m-1} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}_{r,s,t}$, an interaction in $\mathsf{L}_{k,m}$, launching a new $\overline{x}\langle\vec{y}\rangle$, provided $m > 0$.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k-1,m} \,|\, \mathsf{M}_{n,p+1,q} \,|\, \mathsf{N}_{r,s,t}$, a migration of one $\overline{x}\langle\vec{y}\rangle$ into the ether, provided $k > 0$.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p-1,q} \,|\, \mathsf{N}_{r,s+1,t}$, a migration of one $\overline{x}\langle\vec{y}\rangle$ to its target site, provided $p > 0$.
- If $r, t > 0$ then $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n+1,p,q} \,|\, \mathsf{N}_{r-1,s,t-1}$, a migration of one $\overline{e}$ into the ether.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n+1,p,q} \,|\, \mathsf{N}_{r-1,s,t'}$, a migration of one $\overline{e}$ into the ether, given $r > 0$, but $t = 1$.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p+1,q} \,|\, \mathsf{N}_{r,s,t}$, a duplication of one $\overline{x}\langle\vec{y}\rangle$ in the ether, assuming $p > 0$.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n+1,p,q} \,|\, \mathsf{N}_{r,s,t}$, a duplication of one $\overline{e}$ in the ether, assuming $n > 0$.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p-1,q} \,|\, \mathsf{N}_{r,s,t}$, a loss of one $\overline{x}\langle\vec{y}\rangle$ in the ether, assuming $p > 0$.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n-1,p,q} \,|\, \mathsf{N}_{r,s,t}$, a loss of one $\overline{e}$ in the ether, assuming $n > 0$.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}_{r,s,t-1}$, an idle step in the right site, provided $t > 1$.
- $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}_{r,s,t'}$, an idle step in the right site, provided $t = 1$.
- If $s > 0$, the timer can be stopped: $\mathsf{K} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}'_{r,s-1}$.

All of these are matched by the empty transition sequence at either process on the right of the definition of $\mathcal{R}$. We also have two non-τ transitions.

- $\mathsf{K} \xrightarrow{a(\vec{b})}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, \overline{a}\langle\vec{b}\rangle \,|\, \mathsf{N}_{r,s,t-1}$, the import of an output, under the assumption that $a(\vec{b})$ is not H-hidden. It is matched by

  $$[\mathsf{P}\{\vec{y}/\vec{v}\}]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \,|\, \Pi_{i=1}^q \Box_i \xrightarrow{a(\vec{b})}_a [\mathsf{P}\{\vec{y}/\vec{v}\}]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \,|\, \Pi_{i=1}^q \Box_i \,|\, \overline{a}\langle\vec{b}\rangle$$

  and also by

  $$[\star]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \,|\, \Pi_{i=1}^q \Box_i \xrightarrow{a(\vec{b})}_a [\star]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \,|\, \Pi_{i=1}^q \Box_i \,|\, \overline{a}\langle\vec{b}\rangle.$$

- $\mathsf{K} \,|\, \overline{a}\langle\vec{b}\rangle \xrightarrow{\overline{a}\langle\vec{b}\rangle}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}_{r,s,t-1}$, the export of an output, under the assumption that $a(\vec{b})$ is not H-hidden. Here the matches are

  $$[\mathsf{P}\{\vec{y}/\vec{v}\}]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \,|\, \Pi_{i=1}^q \Box_i \,|\, \overline{a}\langle\vec{b}\rangle \xrightarrow{\overline{a}\langle\vec{b}\rangle}_a [\mathsf{P}\{\vec{y}/\vec{v}\}]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \,|\, \Pi_{i=1}^q \Box_i$$

and also

$$[\star]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \,|\, \Pi_{i=1}^q \Box_i \,|\, \overline{a}\langle\vec{b}\rangle \xrightarrow{\overline{a}\langle\vec{b}\rangle}_a [\star]_B^{\mathsf{P}\{\vec{y}/\vec{v}\}} \,|\, \Pi_{i=1}^q \Box_i.$$

The network $[\star]_A^{\overline{x}\langle\vec{y}\rangle \,|\, !e.\overline{x}\langle\vec{y}\rangle} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}_{r,s,t}$ has essentially the same transitions, except that the crashed site can only do two things, idling and restarting. Both are again matched by the empty transition sequence. The networks $\mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, [\star]_B^{\mathsf{P}_{t',t'}}$ and $[\star]_A^{\overline{x}\langle\vec{y}\rangle \,|\, !e.\overline{x}\langle\vec{y}\rangle} \,|\, \mathsf{M}_{n,p,q} \,|\, [\star]_B^{\mathsf{P}_{t',t'}}$ are dealt with in the same way. $\mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}'_{r,s}$ has only one novel transition, namely

$$\mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}'_{r,s} \xrightarrow{\tau}_a \mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, [\Pi_{i=1}^r \overline{e} \,|\, \Pi_{i=1}^s \overline{x}\langle\vec{y}\rangle \,|\, \mathsf{Q}\{\vec{y}/\vec{v}\}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}}$$

But since e, x are fresh, it must be the case that $x \notin \mathsf{fn}(\mathsf{Q}\{\vec{y}/\vec{v}\})$ and we can apply Lemma 61.3 to obtain

$$\mathsf{L}_{k,m} \,|\, \mathsf{M}_{n,p,q} \,|\, [\Pi_{i=1}^r \overline{e} \,|\, \Pi_{i=1}^s \overline{x}\langle\vec{y}\rangle \,|\, \mathsf{Q}\{\vec{y}/\vec{v}\}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}} \quad \approx_a^H \quad [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}}.$$

This means, the empty transition sequence is again a match. The transitions of $[\star]_A^{\overline{x}\langle\vec{y}\rangle \,|\, !e.\overline{x}\langle\vec{y}\rangle} \,|\, \mathsf{M}_{n,p,q} \,|\, \mathsf{N}'_{r,s}$ are matched in the same way. This shows (1). The proof of (2) and (3) is essentially the same. □

The following lemma is very similar to the previous one, except that we add a savepoint. As the save point is "harmless", the proof is also very similar.

LEMMA 65 *Let $A, B \subseteq H$ and x, e fresh. Assume that $0 < t \leq t'$.*

1. *Let $\mathsf{P}_{t,t'} = \mathsf{timer}^t(x(\vec{v}).\mathsf{save}(\mathsf{Q}).\mathsf{Q}, \overline{e} \,|\, \mathsf{P}_{t',t'})$. Then*

$$[\overline{x}\langle\vec{y}\rangle \,|\, !e.\overline{x}\langle\vec{y}\rangle]_A^{\overline{x}\langle\vec{y}\rangle \,|\, !e.\overline{x}\langle\vec{y}\rangle} \,|\, [\mathsf{save}(\mathsf{P}_{t',t'}).\mathsf{P}_{t',t'}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}} \approx_a^H [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}}$$

2. *Let $\mathsf{P}_{t,t'} = \mathsf{timer}^t(x[(\vec{v}).\mathsf{save}(\mathsf{Q}).\mathsf{Q} \;\&\; (\vec{w}).\mathsf{save}(\mathsf{R}).\mathsf{R}], \overline{e} \,|\, \mathsf{P}_{t',t'})$. Then*

$$[\overline{x}\mathsf{left}\langle\vec{y}\rangle \,|\, !e.\overline{x}\mathsf{left}\langle\vec{y}\rangle]_A^{\overline{x}\mathsf{left}\langle\vec{y}\rangle \,|\, !e.\overline{x}\mathsf{left}\langle\vec{y}\rangle} \,|\, [\mathsf{save}(\mathsf{P}_{t',t'}).\mathsf{P}_{t',t'}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}} \approx_a^H [\mathsf{Q}\{\vec{y}/\vec{v}\}]_B^{\mathsf{Q}\{\vec{y}/\vec{v}\}}$$

3. *Let $\mathsf{P}_{t,t'} = \mathsf{timer}^t(x[(\vec{v}).\mathsf{save}(\mathsf{Q}).\mathsf{Q} \;\&\; (\vec{w}).\mathsf{save}(\mathsf{R}).\mathsf{R}], \overline{e} \,|\, \mathsf{P}_{t',t'})$. Then*

$$[\overline{x}\mathsf{right}\langle\vec{y}\rangle \,|\, !e.\overline{x}\mathsf{right}\langle\vec{y}\rangle]_A^{\overline{x}\mathsf{right}\langle\vec{y}\rangle \,|\, !e.\overline{x}\mathsf{right}\langle\vec{y}\rangle} \,|\, [\mathsf{save}(\mathsf{P}_{t',t'}).\mathsf{P}_{t',t'}]_B^{\mathsf{R}\{\vec{y}/\vec{v}\}} \approx_a^H [\mathsf{R}\{\vec{y}/\vec{v}\}]_B^{\mathsf{R}\{\vec{y}/\vec{v}\}}$$

PROOF: The proof of this lemma is essentially the same as that of Lemma 64, except that we must also incorporate the additional state arising from savepointing. This is straightforward, if somewhat tedious. □

Here comes yet another variant of the last two results.

LEMMA 66 *Let $\mathsf{Q}_{t'} = \mathsf{timer}^{t'}(\mathrm{vote}[\overline{c}, \overline{a}], \overline{a})$.*

1. *Abbreviating*

$$\begin{aligned}\mathsf{M} &= [(\nu ca)(\mathsf{Q}_t \mid \mathsf{C}^{\mathrm{and}} \mid \mathsf{C}^{\mathrm{or}})]_A^{\overline{\mathrm{dec}}\mathsf{right} \mid !e.\overline{\mathrm{dec}}\mathsf{right}} \\ \mathsf{N} &= [\overline{\mathrm{vote}}\mathsf{left} \mid \mathsf{P}_t]_B^{\overline{\mathrm{vote}}\mathsf{left} \mid \mathsf{P}_t},\end{aligned}$$

We have $\mathsf{M} \mid \mathsf{N} \approx_a^H [\mathsf{preAbort} \oplus \mathsf{preCommit}]_B^{\mathsf{preAbort} \oplus \mathsf{preCommit}}$.

2. *On the other hand, if*

$$\begin{aligned}\mathsf{M} &= [(\nu ca)(\mathsf{Q}_t \mid \mathsf{C}^{\mathrm{and}} \mid \mathsf{C}^{\mathrm{or}})]_A^{\overline{\mathrm{dec}}\mathsf{right} \mid !e.\overline{\mathrm{dec}}\mathsf{right}} \\ \mathsf{N} &= [\mathsf{save}(\overline{\mathrm{vote}}\mathsf{left} \mid \mathsf{P}_t).(\overline{\mathrm{vote}}\mathsf{left} \mid \mathsf{P}_t)]_B^{\overline{\mathrm{vote}}\mathsf{right} \mid !\overline{\mathrm{abort}}},\end{aligned}$$

we also have $\mathsf{M} \mid \mathsf{N} \approx_a^H [\mathsf{preAbort} \oplus \mathsf{preCommit}]_B^{\mathsf{preAbort} \oplus \mathsf{preCommit}}$.

PROOF: The proof is very similar to that of Lemma 64, except more tedious due to the additional intermediate states induced by the savepoint, and, in case of (2), the additional timer. □

Here is this chapter's version of the Divide-And-Conquer Theorem. It is in some sense easier than that its predecessors because the equational it unifies are more coarse grained.

THEOREM 42 *1. Let* $(\mathsf{M}_i)_{i \in I}$ *be a collection of networks such that for all* $i \in I$*:*

- $\mathsf{ap}(\mathsf{M} \cup A) \subseteq H$*;*
- $\mathsf{M}_i \mid [\mathsf{P}]_A^{\mathsf{P}} \approx_a^H \mathsf{N}$*;*
- $\mathsf{M}_i \mid [\mathsf{Q}]_A^{\mathsf{P}} \approx_a^H \mathsf{N}$*;*
- $\mathsf{M}_i \xrightarrow{l}_a \mathsf{M}$ *where* l *is not* H*-hidden implies* $l = \tau$ *and* $\mathsf{M} \approx_a^H \mathsf{M}_i$*.*

Then $\mathsf{M}_i \mid [\mathsf{P} \oplus \mathsf{Q}]_A^{\mathsf{P}} \approx_a^H \mathsf{N}$.

2. *Let* $(\mathsf{M}_i)_{i \in I}$ *be a collection of networks such that for all* $i \in I$*:*

- $\mathsf{ap}(\mathsf{M}), B \subseteq H$*;*
- $\mathsf{M}_i \equiv \Pi_{j \in I_i} [\mathsf{T}_j]_{A_j}^{\mathsf{U}_j} \mid \Pi_{j \in J_i} [\star]_{A_j}^{\mathsf{U}_j}$*;*
- $\mathsf{M}_i \mid [\mathsf{P}]_A^{\mathsf{P}} \approx_a^H [\mathsf{R}]_B^{\mathsf{R}}$*;*
- $\mathsf{M}_i \mid [\mathsf{Q}]_A^{\mathsf{P}} \approx_a^H [\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}]_B^{\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}}$*; and*
- *if* $\mathsf{M}_i \xrightarrow{l} \mathsf{N}$ *and* l *is not* H*-hidden and not an input, then at least one of the following must be true.*
  - $\mathsf{N} \equiv \mathsf{M}_k$ *for some* $k \in I$*, or*
  - $\mathsf{N} \mid [\mathsf{P} \oplus \mathsf{Q}]_A^{\mathsf{P}} \approx_a^H [\mathsf{R}]_B^{\mathsf{R}}$.

Then for all $i \in I$*:* $\mathsf{M}_i \mid [\mathsf{P} \oplus \mathsf{Q}]_A^{\mathsf{P}} \approx_a^H [\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}]_B^{\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}}$

PROOF: The proof of (1) defines $\mathcal{R}$ up to $\equiv$ by by

$$\mathsf{M}_i \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \,|\, \Pi_{j\in J}\overline{x_j}\langle\vec{y_j}\rangle \;\; \mathcal{R} \;\; \mathsf{N} \,|\, \Pi_{j\in J}\overline{x_j}\langle\vec{y_j}\rangle$$

for all $i \in I$, where each $\overline{x_j}\langle\vec{y_j}\rangle$ is not H-hidden. It is straightforward to see that this defines an asynchronous H-bisimulation, because all computations induced by transitions $\mathsf{M}_i \xrightarrow{l}_a \mathsf{M}$ can be matched by the empty transition sequence at N.

For (2), we define $\mathcal{R}$ up to $\equiv$ by

$$\mathsf{M}_i \,|\, [\mathsf{P}\oplus\mathsf{Q}]^{\mathsf{P}}_A \,|\, \Pi_{j\in J}\overline{x_j}\langle\vec{y_j}\rangle \;\; \mathcal{R} \;\; \begin{cases} [\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}]^{\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}}_B \,|\, \Pi_{j\in J}\overline{x_j}\langle\vec{y_j}\rangle \\ [\star]^{\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}}_B \,|\, \Pi_{j\in J}\overline{x_j}\langle\vec{y_j}\rangle \end{cases}$$

for all $i \in I$. Here each $\overline{x_j}\langle\vec{y_j}\rangle$ is non H-hidden. Note that

$$[\star]^{\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}}_B \xrightarrow{\tau}_a [\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}]^{\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}}_B$$

so we need only match transitions by $\mathsf{M}_i \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \,|\, \Pi_{j\in J}\overline{x_j}\langle\vec{y_j}\rangle$ with transitions from $[\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}]^{\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}}_B \,|\, \Pi_{j\in J}\overline{x_j}\langle\vec{y_j}\rangle$. Now $\mathsf{M}_i \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A$ has the following non H-hidden transitions, where N abbreviates $\Pi_{j\in J}\overline{x_j}\langle\vec{y_j}\rangle$.

- $\mathsf{M}_i \,|\, \mathsf{N} \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \xrightarrow{\tau}_a \mathsf{M}_i \,|\, \mathsf{N} \,|\, [\mathsf{P}]^{\mathsf{P}}_A$. It is matched by

  $$[\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}]^{\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}}_B \,|\, \mathsf{N} \xrightarrow{\tau}_a \xrightarrow{\tau}_a [\mathsf{R}]^{\mathsf{R}}_B \,|\, \mathsf{N}.$$

- $\mathsf{M}_i \,|\, \mathsf{N} \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \xrightarrow{\tau}_a \mathsf{M}_i \,|\, \mathsf{N} \,|\, [\mathsf{Q}]^{\mathsf{P}}_A$. It is matched by the empty transition sequence.

- $\mathsf{M}_i \,|\, \mathsf{N} \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \xrightarrow{\tau}_a \mathsf{M}_i \,|\, \mathsf{N} \,|\, [\star]^{\mathsf{P}}_A$. It is also matched by

  $$[\mathsf{save}(\mathsf{R}).\mathsf{R} \oplus \mathsf{save}(\mathsf{S}).\mathsf{S}]^{\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}}_B \,|\, \mathsf{N} \xrightarrow{\tau}_a \xrightarrow{\tau}_a [\mathsf{R}]^{\mathsf{R}}_B \,|\, \mathsf{N}.$$

- $\mathsf{M}_i \,|\, \mathsf{N} \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \xrightarrow{\tau}_a \mathsf{M} \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A$ because $\mathsf{M}_i \xrightarrow{l}_a \mathsf{M}$, where l is not H-hidden. Then we have two possibilities.

  - $\mathsf{M} \equiv \mathsf{M}_k$ for some $k \in I$, in which case the empty transition sequence matches, or
  - $\mathsf{M} \,|\, [\mathsf{Q}]^{\mathsf{P}}_A \approx^H_a [\mathsf{R}]^{\mathsf{R}}_B$. Then $[\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}]^{\mathsf{save}(\mathsf{R}).\mathsf{R}\oplus\mathsf{save}(\mathsf{S}).\mathsf{S}}_B \,|\, \mathsf{N} \xrightarrow{\tau}_a \xrightarrow{\tau}_a [\mathsf{R}]^{\mathsf{R}}_B \,|\, \mathsf{N}$ is a match.

We have four more transitions which are all induced by (IN), (OUT), (DUPL) and (LOSS) on messages that are not H-hidden.

- $\mathsf{M}_i \,|\, \mathsf{N} \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \xrightarrow{x(\vec{y})}_a \mathsf{M}_i \,|\, \mathsf{N} \,|\, [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \,|\, \overline{x}\langle\vec{y}\rangle$, where $\overline{x}\langle\vec{y}\rangle$ is not H-hidden.

- $\mathsf{M}_i \mid \mathsf{N} \mid \overline{x}\langle\vec{y}\rangle \mid [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{M}_i \mid \mathsf{N} \mid [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A$ ($\overline{x}\langle\vec{y}\rangle$ not H-hidden).
- $\mathsf{M}_i \mid \mathsf{N} \mid \overline{x}\langle\vec{y}\rangle \mid [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{M}_i \mid \mathsf{N} \mid \overline{x}\langle\vec{y}\rangle \mid \overline{x}\langle\vec{y}\rangle \mid [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A$ ($\overline{x}\langle\vec{y}\rangle$ not H-hidden).
- $\mathsf{M}_i \mid \mathsf{N} \mid \overline{x}\langle\vec{y}\rangle \mid [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A \xrightarrow{\tau}_a \mathsf{M}_i \mid \mathsf{N} \mid [\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A$ ($\overline{x}\langle\vec{y}\rangle$ not H-hidden).

They are all matched by doing the corresponding transition on the other side of $\mathcal{R}$. Note that we cannot have a message migration into $[\mathsf{P} \oplus \mathsf{Q}]^{\mathsf{P}}_A$ because by the restrictions on the shape of M_i, all the outputs in the ether are not H-hidden, so their subject cannot be in B. By our requirements, the non H-hidden access points on any two networks related by $\mathcal{R}$ coincide, so $\mathcal{R} \cup \approx_a$ is indeed an asynchronous H-bisimulation. □

The Core of the Proof

We begin as usual with restating the 2PCP in its incarnation with just one participant. First comes the whole protocol.

$$\mathsf{2PCP}_{t,t'} = (\nu vote)(\nu dec)(\nu e)([\mathsf{C}_t]^{\mathsf{C}^{abort}}_{\{vote,e\}} \mid [\mathsf{P}_{t'}]^{\mathsf{P}^{abort}}_{\{dec\}})$$

The next network represents the only participant.

$$\begin{aligned}
\mathsf{P}_t &= \mathsf{P}^{abort} \oplus \mathsf{P}^{precommit}_t \\
\mathsf{P}^{abort} &= \overline{vote}\mathsf{right} \mid !\overline{abort} \\
\mathsf{P}^{precommit}_t &= \mathsf{save}(\mathsf{P}^{commit}_t).\mathsf{P}^{commit}_t \\
\mathsf{P}^{commit}_t &= \overline{vote}\mathsf{left} \mid \mathsf{P}^{wait}_t \\
\mathsf{P}^{wait}_t &= \mathsf{timer}^t(dec[\mathsf{P}^{precommit}, \mathsf{P}^{preabort}], \overline{e} \mid \mathsf{P}^{wait}_t) \\
\mathsf{P}^{precommit} &= \mathsf{save}(!\overline{commit}).!\overline{commit} \\
\mathsf{P}^{preabort} &= \mathsf{save}(!\overline{abort}).!\overline{abort}
\end{aligned}$$

We conclude the presentation of the protocol with the coordinator.

$$\begin{aligned}
\mathsf{C}_t &= \mathsf{C}^{abort} \oplus \mathsf{C}^{commit}_{pre,t} \\
\mathsf{C}^{abort} &= \overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right} \\
\mathsf{C}^{commit}_{pre,t} &= (\nu ca)(\mathsf{C}^{wait}_t \mid \mathsf{C}^{and} \mid \mathsf{C}^{or}) \\
\mathsf{C}^{wait}_t &= \mathsf{timer}^t(vote[\overline{c}, \overline{a}], \overline{a}) \\
\mathsf{C}^{and} &= c.\mathsf{C}^{precommit}_{final} \\
\mathsf{C}^{precommit}_{final} &= \mathsf{save}(\mathsf{C}^{commit}_{final}).\mathsf{C}^{commit}_{final} \\
\mathsf{C}^{commit}_{final} &= \overline{dec}\mathsf{left} \mid !e.\overline{dec}\mathsf{left} \\
\mathsf{C}^{or} &= a.\mathsf{C}^{abort}
\end{aligned}$$

Before we start, we fix the set H of channels that will be restricted when the next two lemmas will be combined.

DEFINITION 74 $H = \{e, dec, vote\}$

The next lemma shows that as soon as there is one decision to abort, the overall decision is also to abort.

LEMMA 67 *Let $t > 0$.*

1. $[\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}^{abort}] \approx_a^H [\mathsf{Abort}]_{\emptyset}^{\mathsf{Abort}}$
2. $[\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}_t^{commit}]^{\mathsf{P}_t^{commit}} \approx_a^H [\mathsf{Abort}]_{\emptyset}^{\mathsf{Abort}}$
3. $[\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}_t^{precommit}]^{\mathsf{P}_t^{abort}} \approx_a^H [\mathsf{Abort}]_{\emptyset}^{\mathsf{Abort}}$
4. $[\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}_t]^{\mathsf{P}_t^{abort}} \approx_a^H [\mathsf{Abort}]_{\emptyset}^{\mathsf{Abort}}$

PROOF: We begin by showing (1).

$$
\begin{aligned}
&[\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}^{abort}]^{\mathsf{P}^{abort}} \\
&\quad \equiv \quad [\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}]^{\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}} \mid [\overline{vote}\mathsf{right} \mid !\overline{abort}]^{\overline{vote}\mathsf{right} \mid !\overline{abort}} \\
&\quad \approx_a^H \quad [\overline{vote}\mathsf{right} \mid !\overline{abort}]^{\overline{vote}\mathsf{right} \mid !\overline{abort}} && (8.1) \\
&\quad \approx_a^H \quad [!\overline{abort}]^{\overline{vote}\mathsf{right} \mid !\overline{abort}} && (8.2) \\
&\quad \approx_a^H \quad [!\overline{abort}]^{!\overline{abort}} && (8.3) \\
&\quad \equiv \quad [\mathsf{Abort}]^{\mathsf{Abort}}
\end{aligned}
$$

Here equation (8.1) follows from Lemma 61.2, while (8.2) is a consequence of Lemma 62.1 and (8.3) of Lemma 62.2.

$$
\begin{aligned}
&[\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}^{commit}]^{\mathsf{P}^{commit}} \\
&\quad \equiv \quad [\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}]^{\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}} \mid [\overline{vote}\mathsf{left} \mid \mathsf{P}_t^{wait}]^{\overline{vote}\mathsf{left} \mid \mathsf{P}_t^{wait}} \\
&\quad \approx_a^H \quad [\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}]^{\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}} \mid [\mathsf{P}_t^{wait}]^{\overline{vote}\mathsf{left} \mid \mathsf{P}_t^{wait}} && (8.4) \\
&\quad \approx_a^H \quad [\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}]^{\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}} \mid [\mathsf{P}_t^{wait}]^{\mathsf{P}_t^{wait}} && (8.5) \\
&\quad \approx_a^H \quad [!\overline{abort}]^{!\overline{abort}} && (8.6)
\end{aligned}
$$

Equation (8.4) is a consequence of Lemma 62.1, (8.5) of Lemma 62.2 and (8.6) of Lemma 64.3. For (3), we proceed almost exactly like for (2).

$$
\begin{aligned}
&[\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \mid [\mathsf{save}(\mathsf{P}^{commit}).\mathsf{P}^{commit}]^{\mathsf{P}^{abort}} \\
&\quad \equiv \quad [\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}]^{\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}} \\
&\qquad \mid [\mathsf{save}(\overline{vote}\mathsf{left} \mid \mathsf{P}_t^{wait}).\overline{vote}\mathsf{left} \mid \mathsf{P}_t^{wait}]^{\overline{vote}\mathsf{right} \mid !\overline{abort}} \\
&\quad \approx_a^H \quad [\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}]^{\overline{dec}\mathsf{right} \mid !e.\overline{dec}\mathsf{right}} \mid [\mathsf{save}(\mathsf{P}_t^{wait}).\mathsf{P}_t^{wait}]^{!\overline{abort}} && (8.7) \\
&\quad \approx_a^H \quad [!\overline{abort}]^{!\overline{abort}} && (8.8)
\end{aligned}
$$

Here (8.7) is a consequence of Lemma 65.3, while (8.8) follows from Lemma 65.2.

For (4), please note that

$$[\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \approx_a^H [!e.\overline{dec}\mathsf{right}]^{\mathsf{C}^{abort}} \mid \Pi_{i=1}^{n}\overline{dec}\mathsf{right}$$

by Lemma 60.2. Hence we can apply Theorem 42.1. □

Next we show, that the coordinator's decision not to abort will mean the overall decision depends on the participant's behaviour.

LEMMA 68 *Let $t > 0$.*

1. $[\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}^{abort}] \approx_a^H [\mathsf{Abort}]_{\emptyset}^{\mathsf{Abort}}$
2. $[\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}_t^{commit}]^{\mathsf{P}_t^{commit}} \approx_a^H [\mathsf{preAbort} \oplus \mathsf{preCommit}]_{\emptyset}^{\mathsf{preAbort}\oplus\mathsf{preCommit}}$
3. $[\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}_t^{precommit}]^{\mathsf{P}_t^{abort}} \approx_a^H [\mathsf{preAbort} \oplus \mathsf{preCommit}]_{\emptyset}^{\mathsf{preAbort}\oplus\mathsf{preCommit}}$
4. $[\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}_t]^{\mathsf{P}_t^{abort}} \approx_a^H [\mathsf{preAbort} \oplus \mathsf{preCommit}]_{\emptyset}^{\mathsf{preAbort}\oplus\mathsf{preCommit}}$

PROOF: We proceed as usual.

$$\begin{aligned}
&[\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}^{abort}} \mid [\mathsf{P}^{abort}] \\
&\quad\equiv\quad [(\nu ca)(\mathsf{timer}^t(vote[\overline{c},\overline{a}],\overline{a}) \mid \mathsf{C}^{and}\mid\ \mathsf{C}^{or})]^{\overline{dec}\mathsf{right}\mid !e.\overline{dec}\mathsf{right}} \\
&\qquad\quad \mid [\overline{vote}\mathsf{right} \mid !\overline{abort}]^{\overline{vote}\mathsf{right}\mid !\overline{abort}} \\
&\quad\approx_a^H\quad [!\overline{abort}]^{\overline{vote}\mathsf{right}\mid !\overline{abort}} && (8.9)\\
&\quad\approx_a^H\quad [!\overline{abort}]^{!\overline{abort}} && (8.10)\\
&\quad\equiv\quad [\mathsf{Abort}]^{\mathsf{Abort}}
\end{aligned}$$

Here (8.9) is a consequence of Lemma 66, while (8.10) can be obtained by application of Lemma 62.3.

The derivation of (2) is essentially Lemma 66.1 while (3) is given by Lemma 66.2.

Finally, for (4), please note that networks in $([\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}^{abort}})_{t>0}$ have only the following non-input, non H-hidden transition.

- $[\mathsf{C}_{pre,t+1}^{commit}]^{\mathsf{C}^{abort}} \stackrel{\tau}{\longrightarrow}_a [\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}^{abort}}$,
- $[\mathsf{C}_{pre,t+1}^{commit}]^{\mathsf{C}^{abort}} \stackrel{\tau}{\longrightarrow}_a [\star]^{\mathsf{C}^{abort}}$
- $[\mathsf{C}_{pre,1}^{commit}]^{\mathsf{C}^{abort}} \stackrel{\tau}{\longrightarrow}_a [(\nu ca)(\overline{c} \mid \mathsf{C}^{and}\mid\ \mathsf{C}^{or})]^{\mathsf{C}^{abort}}$
- $[\mathsf{C}_{pre,t}^{commit}]^{\mathsf{C}^{abort}} \stackrel{\tau}{\longrightarrow}_a [\star]^{\mathsf{C}^{abort}}$

But $[\star]^{\mathsf{C}^{abort}} \approx_a [\mathsf{C}^{abort}]^{\mathsf{C}^{abort}}$ (Lemma 60.1). In addition

$$\begin{aligned}
[(\nu ca)(\overline{a} \,|\, \mathsf{C}^{and} |\, \mathsf{C}^{or})]^{\mathsf{C}^{abort}} &\equiv [(\nu ca)(\overline{a} \,|\, c.\mathsf{C}^{precommit}_{final} \mid \mathsf{C}^{or})]^{\mathsf{C}^{abort}} \\
&\sim_a [(\nu ca)(\overline{a} \,|\, \mathsf{C}^{or})]^{\mathsf{C}^{abort}} \qquad (8.11) \\
&\approx_a [\mathsf{C}^{abort}]^{\mathsf{C}^{abort}} \qquad (8.12)
\end{aligned}$$

Here equation (8.11) follows from Lemma 62.3, while (8.12) is by Lemma 63. Hence we can use Theorem 42.2 to infer (4) from (1) and (3). □

8.5 Concluding Remarks

This chapter has presented the full 2PCP and given a correctness argument. Implementing the algorithm was straightforward, whereas proofs suffered from a proliferation of intermediate states that still awaits taming. Apart from better expansion theorems for message loss and timers in the presence of process failure, it would be nice to have more fine grained reasoning steps for process failure in general. The equational steps taken here seem bigger than necessary. Our optimism that this should be possible comes from the regularity of the behaviour added by process failure: looking at synchronisation trees, it seems that the transition from a model without to one with such failure is just a saturation with certain simple loops, although savepoints may be a bit more complex. In most cases, it should be possible to determine the behaviour of $[\mathsf{save}(\mathsf{P}).\mathsf{P}]^{\mathsf{Q}}_A$ from just knowing what $[\mathsf{Q}]^{\mathsf{Q}}_A$ and $[\mathsf{P}]^{\mathsf{P}}_A$ do.

It would also be good to know more about the map $[\![\cdot]\!]$ which takes processes without savepoints to those that have them, in the way described in §8.2 (by taking savepoint immediately after uncertainty has been removed). We conjecture that $[\![\cdot]\!]$ is fully-abstract and compositional, or at least could be made so by straightforward restrictions on source or target.

Chapter 9

Conclusions and Further Work

We hope to have demonstrated not just that the task is formidable, but also how powerful π-calculi can be as basic building blocks towards a mathematical theory of distributed systems. We take our main accomplishments to be (1) the clean integration of discrete timing with π-calculi, in particular the characterisation of reduction congruence as a labelled bisimilarity; (2) the provision of a simple, yet effective distributed π-calculus with some associated basic reasoning technology; and (3) a straightforward encoding of the 2PCP, a fundamental distributed algorithm. The main benefit of our work is likely to be in its wide range of applicability to problems arising in the construction and verification of DS. Many avenues for further work suggest themselves and some have already been mapped out in previous chapters. In addition, we'd like to mention some more.

Refinement of message and process failure and the associated detection and recovery mechanism would be welcome, but that is unlikely to yield a comprehensive theory of DS. One feature awaiting sustained attention is multicasting, in particular broadcasting, and with those, the enabling message routing infrastructure. The calculus of [34, 35] may be a good starting point.

Security is a big issue in all of computing, but particularly so for DS, if only because their data streams potentially cross many boundaries, conceptual or otherwise. Much problem-specific work has been done [1, 2, 43, 60, 62, 82], but still awaits integration with the kind of DS theory the present work has focused on.

Metalevel or reflexive computation is an area that has spawned a lot of research [71], but little has been assimilated by process theorists. This may be due to a fundamental expressivity gap of name-passing interaction. But then again it might not be. It would be good to see this omission filled in, because DS are plagued by resource management issues that have the flavour of reflexive computation, for example lockable resources and deadlocks.

Finally, given the expressive power of name passing interaction, a natural question, alluded to already, is whether and how added constructs can be represented

in the extended π-calculi. Unfortunately, representability of a given construct in one calculus is not always preserved by adding other computational mechanisms. Consider for example the encoding of branching into the π-calculus [57, 78]:

$$[\![\overline{x}\mathsf{left}\langle\vec{y}\rangle]\!] = (\nu c)(\overline{x}\langle c\rangle \mid c(z_1\, z_2).\overline{z_1}\langle\vec{v}\rangle)$$

(symmetrically for the right selection, and dually for branching input). This encoding is sound and compositional in the base calculus without branching: however the encoding into the full calculus (or the calculus with message loss and timers) is much more involved since we need to consider the case when a branching selection is sent remotely: we either want this interaction sequence implementing the branching output to be lost completely or to be executed safely, in other works, to happen atomically. For this purpose we may be able to use Recursive Timers just as we did for the 2PCP. Thus it may be possible to encode this branching construct into the full calculus, but its representation would be quite complex. This example suggests that individual representability of extensions may not lead to the representability of their combination so that the study of expressiveness for these constructs should be performed with care. Revisiting the accumulated embedding results for distributed π-calculi is likely to result in some surprises.

Bibliography

[1] Martín Abadi and Bruno Blanchet. Secrecy Types for Asymmetric Communication. In *Proc. FoSSaCs'01*, 2001.

[2] Martín Abadi and Andy Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *Proc. ACM Conference on Computer and Communications Security*. ACM Press, 1997.

[3] Parosh Aziz Abdulla, Aurore Annichini, and Ahmed Bouajjani. Algorithmic verification of lossy channel systems. In *Proc. TACAS'99*, 1999.

[4] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Info. & Comp.*, 127(2):91–101, 1996.

[5] Samson Abramsky and Guy McCusker. Game semantics. In *Logic and Computation: Proc. of the 1997 Marktoberdorf Summer School*, 1998.

[6] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[7] Roberto Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous π-calculus. In *Proc. CONCUR'96*, 1996.

[8] Roberto M. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proc. COORDINATION 97*, 1997.

[9] Roberto M. Amadio and Sanjiva Prasad. Localities and failures. In *Proc. FSTTCS'94*, 1994.

[10] Maurice Bach. *The Design of the Unix Operating System*. Prentice-Hall, 1986.

[11] Henk Barendregt. *The Lambda Calculus*. North Holland, 1985.

[12] Martin Berger. (Almost) Every Process Algebra has a Fully-Abstract and Compositional Encoding into the Asynchronous π-Calculus. Draft, 2000.

[13] Martin Berger and Kohei Honda. The Two-Phase Commit Protocol in an Extended π-calculus. Full version of [14]. In preparation.

[14] Martin Berger and Kohei Honda. The Two-Phase Commit Protocol in an Extended π-Calculus. In *Proc. EXPRESS'00*, 2000.

[15] Martin Berger, Kohei Honda, and Nobuko Yoshida. Genericity and the π-Calculus. Draft, 2001.

[16] Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In *Proc. TLCA'01*, 2001.

[17] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[18] Marco Bernardo and Roberto Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *TCS*, 1998.

[19] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[20] Gérard Berry and Gérard Boudol. The Chemical Abstract Machine. *TCS*, 96:217–248, 1992.

[21] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.

[22] Gerard Boudol. Asynchrony and the pi-calculus. Technical Report 1702, INRIA, 1992.

[23] Gérard Boudol. The pi-calculus in direct style. In *Proc. POPL'97*, 1997.

[24] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. In *Proc. EXPRESS'02*, 2002.

[25] Luca Cardelli and Andy Gordon. Mobile ambients. *TCS*, 240, 2000.

[26] Ilaria Castellani. Process algebras with localities. In [17], chapter 15. North-Holland, 2001.

[27] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000.

[28] B. Jack Copeland. The Church-Turing Thesis. Entry in the Stanford Encyclopedia of Philosophy, http://plato.stanford.edu/entries/church-turing/.

[29] B. Jack Copeland and R. Sylvan. Beyond the universal turing machine. *Australasian Journal of Philosophy*, 1998.

[30] George Couloris, Jean Dollimore, and Tim Kindberg. *Distributed Systems, Concepts and Design.* Addison-Wesley, 2001.

[31] Martin Davis and Elene Weyuker. *Computability, Complexity, and Languages.* Academic Press, 1983.

[32] Pierpaolo Degano, Jean-Vincent Loddo, and Corrado Priami. Mobile processes with local clocks. In *Proc. of the Workshop on Analysis and Verification of Multiple-Agent Languages*, 1996.

[33] Elmootazbellah N. Elnozahy, David B. Johnson, and Yi-Min Wang. A survey of rollback-recovery protocols in message-passing systems. Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, 1996.

[34] Cristian Ene and Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In *Proc. FCT'99*, LNCS, 1999.

[35] Cristian Ene and Traian Muntean. A Broadcast-Based Calculus for Communicating Systems. In *Workshop on Formal Methods for Parallel Programming*, 2001.

[36] Lindsay Errington. *Twisted Systems.* PhD thesis, Imperial College, 1999.

[37] Cédric Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming.* PhD thesis, Ecole Polytechnique, 1998.

[38] Cédric Fournet and Georges Gonthier. A hierarchy of equivalences for asynchronous calculi. In *Proc. ICALP'98*, 1998.

[39] Cédric Fournet, Georges Gonthier, Jean-Jaques Lévy, Luc Maranget, and Didier Rémy. A Calculus of Mobile Agents. In *Proc. POPL'96*, 1996.

[40] Simon J. Gay. A sort inference algorithm for the polyadic pi-calculus. In *Proc. POPL'93*, 1993.

[41] Jean-Yves Girard. Linear logic. *TCS*, 50, 1987.

[42] Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *MSCS*, 11(3), 2001.

[43] Andy Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. In *Proc. CSFW 2002*, 2002.

[44] Eric Goubault. *The Geometry of Concurrency.* PhD thesis, École Polytechnique, 1995.

[45] Eric Goubault. Geometry and concurrency: A user's guide. *MSCS*, 10(4), 2000.

[46] Ronald Graham, Oren Patashnik, and Donald E. Knuth. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1994.

[47] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Morgan Kaufmann, 1993.

[48] Carl A. Gunter. *Semantics of Programming Languages*. MIT Press, 1995.

[49] Matthew Hennessy. Timed process algebras: a tutorial, 1992.

[50] Matthew Hennessy and Robin Milner. Algebraic Laws for Non-Determinism and Concurrency. *JACM*, 32(1), 1985.

[51] Rolf Herken, editor. *The Universal Turing Machine: a Half-Century Survey*. Springer, 1995.

[52] Tony Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[53] Kohei Honda. Types for the π-calculus. http://www.dcs.qmul.ac.uk/~kohei.

[54] Kohei Honda. Two bisimilarities in ν-calculus. Technical Report 92-002, Keio University, Department of Computer Science, 1992.

[55] Kohei Honda. Types for dyadic interaction. In *Proc. CONCUR '93*, number 715 in LNCS, pages 509–523. Springer, 1993.

[56] Kohei Honda. Elementary Structures for Process Theory (1): Sets with Renaming. *MSCS*, 2001.

[57] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *Proceedings of ECOOP'91*, 1991.

[58] Kohei Honda and Mario Tokoro. A small calculus for concurrent objects. *OOPS Messenger*, 2(2):50–54, 1991.

[59] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. *TCS*, 151, 1995.

[60] Kohei Honda, Vasco T. Vasconcelos, and Nobuko Yoshida. Secure information flow as typed process behaviour. In *Proc. ESOP'99*, 2000.

[61] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *TCS*, 151, 1995.

[62] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. In *POPL'02*, 2002.

[63] Winfried Just and Martin Weese. *Discovering Modern Set Theory.* AMS, 1996.

[64] Marco Kick. Bialgebraic Modelling of Timed Processes. In *Proc. ICALP'02*, 2002.

[65] Marco Kick. Rule Formats for Timed Processes. In *Proc. CMCIM'02*, 2002.

[66] Kenneth Kunen. *Set Theory: An Introduction to Independence Proofs.* North-Holland, 1980.

[67] James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *Proc. CONCUR'2000*, 2000.

[68] Giuseppe Longo. The difference between clocks and turing machines. In Arturo Carsetti, editor, *Functional Models of Cognition.* Kluwer, 1999.

[69] Luís Miguel Barros Lopes. *On the Design and Implementation of a Virtual Machine for Process Calculi.* PhD thesis, Departamento de Cincia de Computadores, FCUP, 1999.

[70] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[71] Narciso Martí-Oliet and José Meseguer. Rewriting Logic: Roadmap and Bibliography. *TCS*, 2001.

[72] Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. In *Proc. ICALP'98*, 1998.

[73] Dave Mills. Time synchronization server. URL http://www.eecis.udel.edu/~ntp/.

[74] Robin Milner. *Communication and Concurrency.* Prentics Hall, 1989.

[75] Robin Milner. Functions as processes. *MSCS*, 2(2):119–141, 1992.

[76] Robin Milner. Elements of interaction. *CACM*, 36(1), 1993.

[77] Robin Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, 1999.

[78] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Info. & Comp.*, 100(1), 1992.

[79] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *Proc. ICALP'92*, 1992.

[80] Yiannis N. Moschovakis. *Notes on Set Theory.* Springer, 1994.

[81] Sape Mullender, editor. *Distributed Systems.* Addison-Wesley, 1993.

[82] Andrew C. Myers and Andrei Sabelfeld. Language-based information-flow security. *IEEE Journal on Selected Areas in Communication*, 2002.

[83] Roger Needham. Names. In [81], pages 315–327. Addison-Wesley, 1993.

[84] Uwe Nestmann. What is a 'good' encoding of guarded choice? Technical report, BRICS, 1997.

[85] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. In *Proc. CONCUR'96*, 1996.

[86] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part i. *TCS*, 13(1):85–108, 1981.

[87] Piergiorgio Odifreddi. *Classical Recursion Theory.* North-Holland, 1989.

[88] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proc. POPL*, 1997.

[89] Christos H. Papadimitriou. *Complexity Theory.* Addison Wesley, 1994.

[90] Benjamin C. Pierce. *Type Systems and Programming Languages.* MIT Press, 2002.

[91] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner.* MIT Press, 2000.

[92] Gordon D. Plotkin. A structural approach to operational semantics. Technical report, DAIMI, Aarhus University, 1981.

[93] Paola Quaglia and David Walker. On encoding pπ in mπ. In *Proc. 18th FST & TCS*, 1998.

[94] Wolfgang Reisig. *Petri Nets, an Introduction.* Springer, 1985.

[95] Wolfgang Reisig and Grzegorz Rozenberg, editors. *Lectures on Petri Nets I: Basic Models.* Springer, 1998.

[96] James Riely and Matthew Hennessy. Resource access control in systems of mobile agents. *ENTCS*, 16(3), 1998.

[97] James Riely and Matthew Hennessy. A typed language for distributed mobile processes. In *Proc. POPL 1998*, 1998.

[98] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. In *Proc. POPL 1999*, 1999.

[99] James Riely and Matthew Hennessy. Distributed processes and location failures. *TCS*, to appear.

[100] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms*. PhD thesis, University of Edinburgh, 1992.

[101] Davide Sangiorgi. Internal mobility and agent passing calculi. In *Proc. ICALP'95*, 1995.

[102] Davide Sangiorgi. πI: A symmetric calculus based on internal mobility. In *Proc. TAPSOFT'95*, 1995.

[103] Davide Sangiorgi and David Walker. *The π-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[104] Davide Sangiorgi and David Walker. Some results on barbed equivalences in pi-calculus. In *Proc. CONCUR'01*, 2001.

[105] Ichiro Satoh and Mario Tokoro. Semantics for a real-time object-oriented programming language. In *Proc. ICCL'94*, 1994.

[106] Peter Selinger. First-order axioms for asynchrony. In *Proc. CONCUR '97*, 1997.

[107] Peter Sewell. From rewrite rules to bisimulation congruences. In *Proc. CONCUR 98*, 1998.

[108] Peter Sewell. Global/local subtyping and capability inference for a distributed pi-calculus. In *Proc. ICALP'98*, 1998.

[109] Cosma Rohilla Shalizi. *Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata*. PhD thesis, University of Wisconsin, 2001.

[110] Saharon Shelah. Every two elementary equivalent models have isometric ultrapowers. *Israel Journal of Mathematics*, 10:224–233, 1971.

[111] Andrew S. Tannenbaum. *Computer Networks*. Prentice Hall, 1996.

[112] Bent Thomsen. Plain CHOCS: A Second Generation Calculus for Higher Order Processes. *Acta Informatica*, 30(1):1–59, 1993.

[113] David N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation.* PhD thesis, University of Edinburgh, 1996.

[114] Vasco Vasconcelos. Typed concurrent objects. In *Proc. ECOOP'94*, 1994.

[115] Vasco T. Vasconcelos and Kohei Honda. Principal typing scheme for polyadic π-calculus. In *Proc. CONCUR'93*, 1993.

[116] Vasco T. Vasconcelos and António Ravara. Communication errors in the pi-calculus are undecidable. *Information Processing Letters*, 71(5–6):229–233, 1999.

[117] Jan Vitek and Giuseppe Castagna. Seal: A Framework for Secure Mobile Computations. In *Internet Programming Languages*, 1999.

[118] Glynn Winskel. Event structures. In *Advances in Petri Nets 1986, Part II*, 1987.

[119] Glynn Winskel. *The formal semantics of programming languages.* MIT Press, 1993.

[120] Glynn Winskel and Mogens Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science.* Oxford University Press, 1995.

[121] Paweł Wojciechowski. *Nomadic Pict: Language and Infrastructure Design for Mobile Computation.* PhD thesis, University of Cambridge, 2000.

[122] Wang Yi. Real-time behaviour of asynchronous agents. In *Proc. CONCUR'90*, pages 502–520, 1990.

[123] Nobuko Yoshida. Graph Types for Monadic Mobile Processes. In *Proc. FSTTCS'96*, 1996.

[124] Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong Normalisation in the π-Calculus. In *Proc. LICS'01*, 2001.