To appear as Tech Report

Pi Implementation (2): A Reliable Protocol for Synchronous Rendezvous

Lucian Wischik **Damon Wischik**

January 2004

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in gzipped PostScript format via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`. All local authors can be reached via e-mail at the address *last-name*`@cs.unibo.it`.

Recent Titles from the UBLCS Technical Report Series

2002-9 *Towards Adaptive, Resilient and Self-Organizing Peer-to-Peer Systems*, Montresor, A., Meling, H., Babaoglu, O., September 2002.

2002-10 *Towards Self-Organizing, Self-Repairing and Resilient Distributed Systems*, Montresor, A., Babaoglu, O., Meling, H., September 2002 (Revised November 2002).

2002-11 *Messor: Load-Balancing through a Swarm of Autonomous Agents*, Montresor, A., Meling, H., Babaoglu, O., September 2002.

2002-12 *Johanna: Open Collaborative Technologies for Teleorganizations*, Gaspari, M., Picci, L., Petrucci, A., Faglioni, G., December 2002.

2003-1 Security and Performance Analyses in Distributed Systems (Ph.D Thesis), Aldini, A., February 2003.

2003-2 Models and Types for Wide Area Computing. The calculus of Boxed Ambients (Ph.D. Thesis), Crafa, S., February 2003.

2003-3 MathML Formatting (Ph.D. Thesis), Padovani, L., February 2003.

2003-4 Performance Evaluation of Mobile Agents Paradigm for Wireless Networks (Ph.D. Thesis), Al Mobaideen, W., March 2003.

2003-5 Synchronized Hypermedia Documents: a Model and its Applications (Ph.D. Thesis), Gaggi, O., March 2003.

2003-6 Searching and Retrieving in Content-Based Repositories of Formal Mathematical Knowledge (Ph.D. Thesis), Guidi, F., March 2003.

2003-7 Intersection Types, Lambda Abstraction Algebras and Lambda Theories (Ph.D. Thesis), Lusin, S., March 2003.

2003-8 Towards an Ontology-Guided Search Engine, Gaspari, M., Guidi, D., June 2003.

2003-9 An Object Based Algebra for Specifying A Fault Tolerant Software Architecture, Dragoni, N., Gaspari, M., June 2003.

2003-10 A Scalable Architecture for Responsive Auction Services Over the Internet, Amoroso, A., Fanzieri F., June 2003.

2003-11 WSSecSpaces: a Secure Data-Driven Coordination Service for Web Services Applications, Lucchi, R., Zavattaro, G., September 2003.

2003-12 Integrating Agent Communication Languages in Open Services Architectures, Dragoni, N., Gaspari, M., October 2003.

2003-13 Perfect load balancing on anonymous trees, Margara, L., Pistocchi, A., Vassura, M., October 2003.

2003-14 Towards Secure Epidemics: Detection and Removal of Malicious Peers in Epidemic-Style Protocols, Jelasity, M., Montresor, A., Babaoglu, O., November 2003.

2003-15 Gossip-based Unstructured Overlay Networks: An Experimental Evaluation, Jelasity, M., Guerraoui, R., Kermarrec, A-M., van Steen, M., December 2003.

2003-16 Robust Aggregation Protocols for Large-Scale Overlay Networks, Montresor, A., Jelasity, M., Babaoglu, O., December 2003.

Pi Implementation (2): A Reliable Protocol for Synchronous Rendezvous

Lucian Wischik **Damon Wischik**

Technical Report

January 2004

Abstract

In the presence of failure, any protocol for distributed atomic commitment will have certain unavoidable limitations. These limitations turn out not so serious in one special case – that of synchronous rendezvous. Rendezvous is important because it is the basis for process calculi, which themselves underpin several new experimental languages and also web services.

We give a simplified three phase commit protocol specially tailored to rendezvous. In the presence of arbitrary message loss and permanent site failure, the protocol is strongly non-blocking for one party – the party can always unblock immediately. This is useful for writing a reliable non-blocking web service. If message loss is fair and site failure is not permanent, then the protocol is also weakly non-blocking for the other party – the chance of it remaining blocked tends to zero as time increases. (This yields a solution to the classic 'Two Generals' problem, which is a degenerate case of rendezvous).

The proof of non-blocking uses a novel technique involving Markov processes. It is a general technique that applies to any calculus and any implementation with message loss, so long as the two are bisimilar.

1 Introduction

Process calculi with rendezvous [13, 20] have been used as the basis for experimental concurrent programming languages [5, 10, 15, 23, 32], and are also being used for emerging standards in web services [2, 9, 28]. Transactions have a long history in database research. When process calculi meet with transactions [3, 7, 6, 8, 21] they provide new motivations, new problems, and new theoretical techniques. The new theoretical techniques are precise mathematical formalisms, with *bisimulation* to prove correctness over all possible execution traces. (Transaction *validity* is simulation; transactional *non-triviality* and *weak termination* are special cases of inverse simulation, and bisimulation is the combination of the two simulations.) The new problem involves *synchronous rendezvous* – a special case of transactions, with interesting properties.

Synchronous rendezvous is a communication primitive for multithreaded and distributed programming. It assumes a distributed set of communication channels. One or more machines signal their wish to send data over a channel, and they then block. Other machines signal their wish to receive data over a channel, and also block. When a sender/receiver pair match up, they exchange data and both then unblock. This is atomic: either both parties in the pair unblock, or neither does and both remain available for other pairings. Synchronous rendezvous is the basic mechanism of computation in the pi calculus [20], a canonical calculus of concurrency.

Synchronous rendezvous is a special case of distributed atomic transaction. Normally, a transaction requires its participants to either all abort, or all commit. One kind of transaction called a *cohesion* [22] has lesser requirements: either all abort, or a subset commit while the rest abort. Rendezvous is a 'binary' cohesion – it requires that exactly one sender and one receiver commit to interact while all other participants abort. (There is also an additional requirement of rendezvous, *freedom*, discussed below).

There is a standard impossibility result for atomic commitment [12, 27]: in the presence of failure, at least some committing participants may be forced to block until they can re-establish communication with the others. Rendezvous is still subject to this impossibility result, but it is less serious – one party to the rendezvous (such as a web service) will never block, while the other party (a client that connects to it) might. Through exploiting the binary nature of rendezvous, we obtain this result with a three phase commit protocol that is shorter than the version traditionally used. In particular, the traditional protocol has one coordinator and several participants; in ours, the offering party is coordinator for the first phase and the accepting party is coordinator for the next two. The traditional protocol uses an extra quorum-based recovery phase in the presence of message loss [26]; ours does not, since (in binary rendezvous) the accepting party alone is already quorate.

The standard 'Two Generals' problem [18] is a special case of synchronous rendezvous. The problem has two Roman armies at the top of two different hills, with barbarians in the valley. The generals communicate via messengers, who may be intercepted by the barbarians. If both generals decide to attack then the barbarians will be crushed; if only one attacks then the barbarians will win. In rendezvous terms, one program wishes to send on a channel and one wishes to receive on it. Either both programs commit to interacting and so unblock, or neither one does. It is commonly (but imprecisely) stated that the two generals problem has no solution. More precisely, it has no solution without an unbounded number of messages. The results presented here are consistent with that more precise statement, but can be stated positively: *the two generals problem has a solution which never yields a lone attack, and which yields a unified attack with a probability that tends to 1 as time increases*. In computer science such a property is called 'divergence'; in modal logic it is called 'eventuality' and written $\diamond$; in probability theory the result is said to be 'almost certain' or 'overwhelmingly likely'. Alas for the generals, it only helps them agree in principle, rather than agree to a fixed time. We return to the Two Generals problem in the conclusion.

There is one additional requirement of rendezvous, 'freedom': *If there exists a send/receive pair where both parties are uncommitted or aborted, then rendezvous between this pair is possible.* (This is strictly stronger than conventional non-triviality). For a concrete example of freedom after an abort, consider the proposed rendezvous between Romeo and Juliet [25]. Tragically a confirmation message from Juliet is lost, and so Romeo aborts this rendezvous and chooses instead to

go with the Poison. Because of the abort, Juliet will not keep waiting in vain for Romeo, but is instead free to choose other partners such as the Sword. A happier example of freedom is given in Figure 4 (page 20).

In the process calculi field synchronous rendezvous has not been widely used, because it has not been clear how to achieve it reliably in the presence of failures. Instead, most work has focused on *asynchronous* rendezvous [1]. This is where the sending party does not block, and indeed does not know whether the message was successfully delivered. Such an asynchronous rendezvous is good as a model for a fallible network, but it makes for an obnoxious primitive in a practical programming language – akin to not knowing whether a system call succeeded, and not being able to check its return code!

If there were no failures, it would be easy to implement synchronous rendezvous just in terms of asynchronous: the receiver is programmed to send back an acknowledgement, and the sending party blocks until it has received this acknowledgement. (This two-phase rendezvous is a degenerate case of the trivial two-phase solution to distributed transactions in the absence of failure). But consider now what happens if there is communication failure – the sending party blocks for ever, waiting in vain for the acknowledgement. The cause of this wait is *either* that the acknowledgement has been lost, *or* that the original send has been lost. In an attempted solution the receiver party might be programmed to keep sending an acknowledgement until at least one gets through, but this does not help if the original 'send' was lost. The sending party might be programmed to keep sending its 'send' to all possible receivers until at least one gets through, but this yields an error if two sends got through. In effect the sending party *needs* to know whether its send got through – ie. it needs something like synchronous rendezvous!

In the asynchronous *localised* case [13, 14] a simpler halfway solution is possible. Localised means that receivers are all at the same location, and might collectively be thought of as comprising just a single partner in the transaction. (Equivalently, receivers are never partitioned amongst themselves, and no receiver ever fails unless they all do.) This means that the freedom requirement of possible rendezvous between *any* pair becomes degenerate – there is only one pair – and a blocking protocol such as two-phase can be used. In particular, for encoding synchronous into asynchronous, the send command becomes 'keep trying to send until I hear that it worked', and similarly the acknowledge becomes 'keep trying to acknowledge until I hear that it worked'. This is equivalent to simply using a reliable messaging transport, as is done with Microsoft Biztalk [11]. However, many real-world situations are non-localised, such as buying an airline ticket from one of several different companies or signing onto one of two different game servers. Therefore, even in Biztalk and asynchronous calculi, one must still implement some synchronous protocol such as the one described here.

Our approach is indeed to implement rendezvous transactions in an asynchronous process calculus formalism. Other, simpler transactions have been implemented this way before. Notably, in [3] there is an implementation of the standard two phase commit protocol in an asynchronous process calculus, with proofs of 'eventual' correctness in the presence of failures. We remark that two phase commit assumes a fixed set of participants, and so does not give the *freedom* required by rendezvous. That is why our current work instead uses a three phase protocol – one which does give freedom. (Although [3] uses timers for its protocol, it could achieve the same 'eventual' correctness result merely through non-determinism, as is done here.) Further work on transactions and process calculi [7, 6, 8] has instead not dealt with failure. Some recent work [21] gives a process calculus formalism and proof for a distributed consensus algorithm in the presence of failures, although consensus again uses a fixed set of participants. (This algorithm also uses 'magical failure detectors', based on a weaker failure model than the arbitrary message loss we assume for the current work.)

In summary, the possibility of failure *requires* synchronous rendezvous: as a programming primitive it is far preferable to the asynchronous alternative; and even in an asynchronous language, a reliable synchronous rendezvous still has to be programmed eventually.

Romeo should have used a three-phase commit protocol

Our original contribution in this paper is to give a new three phase commit protocol for implementing synchronous rendezvous. We prove it correct with respect to the pi calculus, a paradigmatic calculus for concurrency and synchronous rendezvous. In particular, we prove that the protocol is a faithful implementation of the calculus – the implementation admits a rendezvous if and only if the calculus does, even in the presence of failures.

We use three models for failures, each progressively more specific, and give progressively stronger results for each:

1. Assume *communication failure*, also called arbitrary message loss. We prove that if the implementation can make a particular interaction, then so can the calculus. This property is known as *validity*. We also prove that if the calculus can make a particular interaction, then (if undesired failure does not happen) so can the implementation. This result combines *weak termination* and *freedom*, which is strictly stronger than *non-triviality*. Together, the two results amount to *bisimulation* between calculus and implementation.

2. Assume that message loss is *fair*: messages may fail as above, but will not perpetually fail. We prove that the protocol is 'overwhelmingly likely' non-blocking: the chance of remaining blocked tends to zero as time increases. The Two Generals problem uses the fair failure model, and so it too admits a solution that never terminates incorrectly and is 'overwhelmingly likely' to terminate correctly; equivalently, it admits a solution which 'eventually' succeeds.

3. Assume *site failure*: all messages to or from a site perpetually fail, presumably because a device has crashed or been turned off. We exhibit a type system which guarantees that a server in communication with such a device will not block.

In this paper, for simplicity, we consider a broadcast network. This simplifies the task of discovery (ie. discovering which other parties on the network are willing to respond to our request); it also allows a trivial implementation of distributed choice. However, the protocol seems suitable also for a point-to-point network. We remark that a point-to-point network is much like a fallible broadcast network in which each broadcast message fails to reach all but one recipient. Also for simplicity we include no explicit notation of locations or for the failure of a location. Instead, we model such a failure as the failure of all messages to or from a group of processes.

Our proof for non-blocking uses a novel technique involving Markov processes (Section 5). It is a general technique that adapts straightforwardly to any calculus and any implementation with failures, so long as the two are bisimilar.

We stress that although we assume an asynchronous, broadcast, unreliable network model, the pi calculus that we implement on top of it is synchronous, point-to-point and reliable.

2 Protocol

We now give an informal description of the protocol for synchronous rendezvous. The protocol is drawn as a finite state machine in Figure 1, some example message-traces are given in Figure 2, a sketch is given in Figure 4, and the protocol is presented formally in the following section.

Overview: the protocol first picks a possible pair that is interested in rendezvous together, and then tries to make the rendezvous happen. If no possible pair was found, or if the rendezvous didn't work out, then the system is left in the same state it was initially – free to pick another possible pair and start again. The initial pick is lightweight and does not attempt (or need) to deal with failures. But once a pick has been made, then the pair make a decision about whether to commit or abort this particular pick, and their decision is communicated reliably to each other. What follows gives more details to this overview:

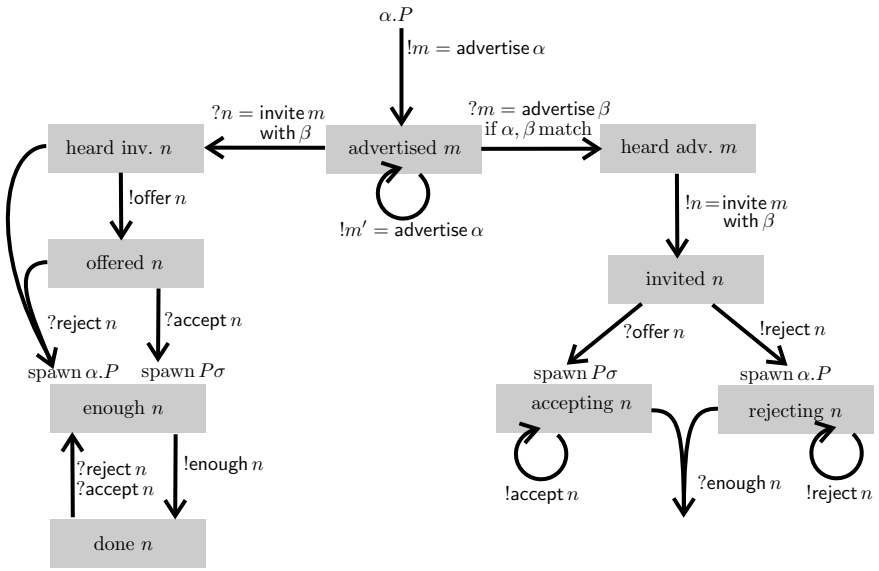


Figure 1: The protocol for rendezvous, as a finite state machine. The initial term $\alpha.P$ may be an input or output command. It may end up initiating a rendezvous (left branch) or responding to a rendezvous (right branch). The *spawn* command starts another fragment running in parallel.

1. Any party with an output command can, at any time, *advertises* this fact and block. It generates a globally unique 'transaction identifier' for each advertisement it makes. (We assume this identifier for the following steps of the protocol). If it has an advertisement outstanding, it can supersede it by broadcasting a different advertisement.

2. Any party with an input command who hears the advertisement can *invite* the advertiser to make an offer. (For efficiency, if it heard someone else broadcast an invitation first, then it might choose not to.) The input party blocks.

3. The output party will hear zero or more invitations. If it hears one or more, and has not yet superseded its advertisement, then it chooses one of them, and broadcasts back an *offer*.

4. While it is waiting for an offer, the input party can non deterministically choose to abort. It broadcasts an *reject* message, and goes on broadcasting it. Eventually the output party will receive the message, and at this point it also knows to abort. When it receives the message, it replies with *enough*. Subsequently, each time it hears another *reject*, it again replies with *enough*. When the input party eventually hears *enough*, it stops sending *reject*.

5. On the other hand, if the input party did receive an offer and did not yet choose to abort, then all is well: it unblocks, and broadcasts an *accept* message, and goes on broadcasting it. When the output party hears this message it also unblocks. Again it replies *enough*, and keeps replying to *accept* with *enough* until the *accept* stops coming.

6. (The exact symmetric case is also possible, for when the input command broadcasts an advertisement, and an output command invites offers. A party with an outstanding advertisement can supersede it by responding to some other advertisement.)

This is a three phase commit protocol: first (step 1) the advertiser waits for invitations; second (steps 2,3) the inviter waits for offers; third (steps 4,5) the advertiser waits for *accept* or *reject*. We give an approximate translation from our notation to traditional three phase commit notation, as found in [4]. The translation is only approximate, since our protocol effectively involves parallel

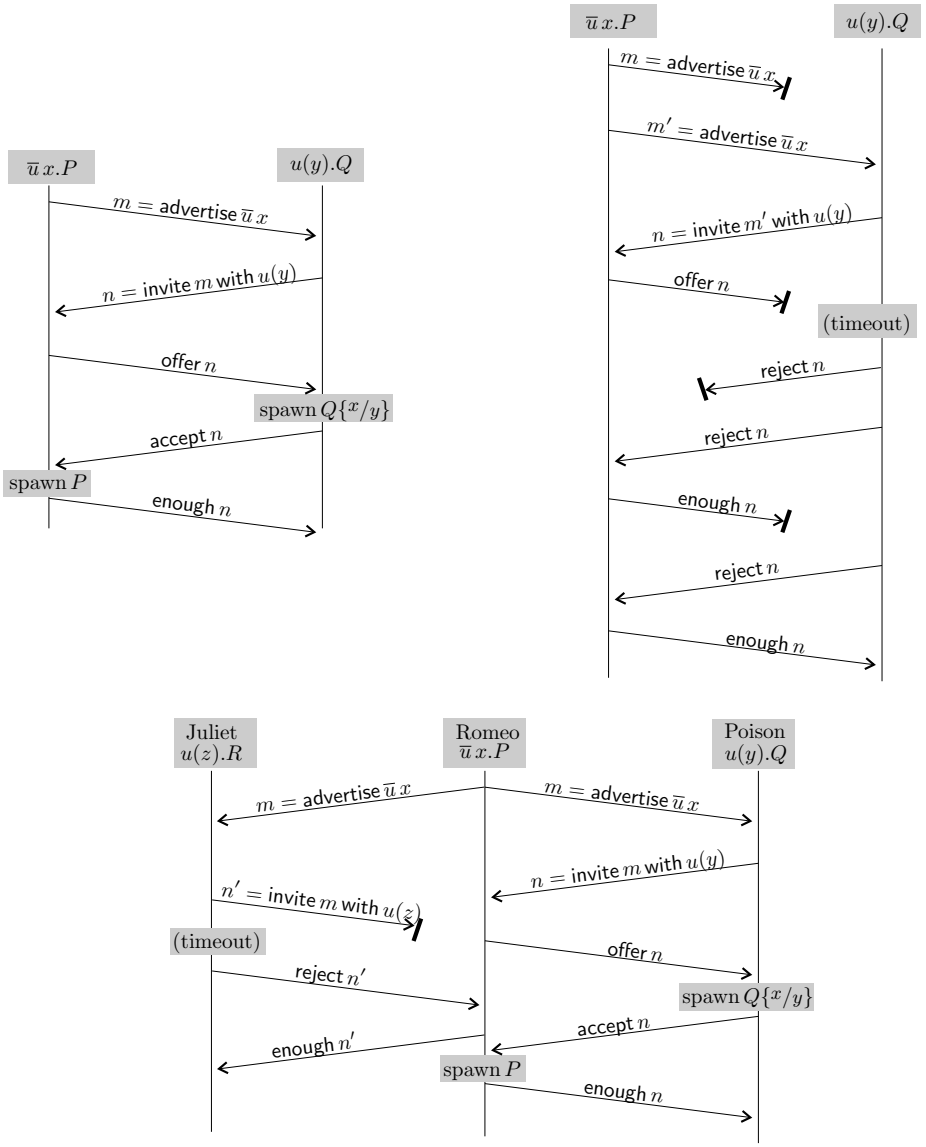


Figure 2: Some instances of the synchronous-rendezvous protocol. (Top-left) A simple two-party interaction with no failures. (Top-right) An attempted two-party interaction, but aborted due to too many failures. (Bottom) Juliet's message is lost, so Romeo chooses the Poison instead.

transactions (one for the offer, one for the acceptance), as suggested by its use of two transaction identifiers.

rendezvous protocol	*advertiser as coordinator*	*inviter as coordinator*
advertise message	VOTE-REQ	YES
invite message	YES	PRE-COMMIT
offer message	.	ACK
accept message	.	COMMIT

In the traditional three phase commit protocol, the advertiser is the *coordinator* of the transaction. It sends VOTE-REQ, and everyone votes YES or NO. If anyone voted NO then the coordinator tells everyone to ABORT. If everyone voted YES then it tells everyone to PRE-COMMIT, so everyone ACKs, and the coordinator collects ACKs from all other participants. When it has received them all, it unblocks and sends COMMIT to them all, whereupon they also unblock.

In our version, the advertiser only coordinates the first phase of the protocol – as soon as another party invites offers, then this inviter becomes itself the coordinator of a second transaction. The earlier advertisement already counts as a 'precocious' YES vote (albeit one that can be later changed by withholding ACK). The inviter can therefore continue immediately with PRE-COMMIT and the rest of the transaction. This is however only an approximation. In the tradition protocol, if no ACK is received then the coordinator commits anyway. In our protocol, if no *offer* is received then the inviter instead aborts.

Even in the presence of communication failure, our protocol works as it is. But the traditional protocol instead uses a more complicated recovery-process: it attempts to form a quorum, so that commitment is possible when at least a majority of participants know the intended outcome [26]. The difference between our protocol and this quorum may be understood thus: rendezvous is between two parties only, so we asymmetrically define the accepting party to be quorate even alone, and so the quorum-formation algorithm is not relevant.

Consider the rendezvous requirement of *freedom* and how it relates to the protocol. Freedom is the property that, if there is *any* uncommitted send/receive pair, it should be possible for it to rendezvous. (This is a stronger version of non-triviality). To achieve this, the first phase/transaction of our protocol searches amongst *all* parties who might potentially be interested in rendezvous on a particular channel. If this first phase were done with a blocking protocol such as two phase commit, then freedom would be violated.

Although our protocol has not been published before, it is not all that different from existing practice. Plumbers might advertise their services in a telephone directory. A homeowner invites them to offer bids. They make their offers. The homeowner accepts one. (In many countries, including Italy, America and England, the contract is deemed to be formed at the moment the acceptance was put in a letter-box, even if it is subsequently lost in the post. Other legalities are different – in the England a 'notice to quit' takes effect only when the letter is received; in Italy it takes effect only when registered by a centralised government service.)

We remark upon an implementation detail for the *enough* messages. The globally unique transaction identifier n will actually incorporate the identity of one particular machine (location) such that all interactions involving the identifier will also involve programs on that particular machine. Hence, contrariwise, if that same machine hears a message on n that none of its current programs can handle, then the machine itself can reply *enough* n. Specifically, this implementation involves the message '$!m =$invite n with a' in Figure 1: the same machine that generated n will also be responsible for all *enough* m messages.

This implementation mechanism involves machines (locations). However we have avoided locations in our current formalism for reasons of simplicity. Instead we have approximated the mechanism, as we now explain with reference to Figure 1. We introduced a state *enough* m whose only job is to generate *enough* messages. This state might be called a 'zombie' state since it is left behind after each attempted interaction and has no other use. It is not needed in the implementation. There is additionally one unusual case, where a party invites but its invitation is lost. Here the zombie will not be generated in the formalism, so the party has no one to tell it *enough*, and so it perpetually sends *abort*. The formalism is resilient enough to cope with this case – but

actually in the implementation there will always be someone to say *enough*, as per the previous paragraph.

In the failure scenario of machines crashing, if a machine crashes part way through an interaction, then another party might send a stream of *reject* or *accept* messages until the machine is rebooted. We could perhaps bound the size of this stream by sending each message after twice the delay of the previous. More specific solutions depend on precisely what kind of logging the machine used (and hence, the state to which it restores itself after a crash).

3 Formalism

We now present the protocol formally. Assume an infinite set of channel-names $x, y, \ldots$ and of message-identifiers $n, m, \ldots$.

Definition 1 (Syntax) *Programs P, atoms A and machines M are*

$$\begin{array}{lcl}
\alpha & ::= & \overline{u}\,\widetilde{x} \;\mid\; u(\widetilde{x}) \\
P & ::= & \alpha.P \;\mid\; !\alpha.P \;\mid\; \nu x.P \;\mid\; P|P \;\mid\; \mathbf{0} \\
A & ::= & \mathsf{adv}\, n.\alpha.P \;\mid\; \mathsf{hinv}\, n.\sigma.\alpha.P \;\mid\; \mathsf{off}\, n.\sigma.\alpha.P \;\mid\; \mathsf{hadv}\, n.\sigma.\alpha.P \\
 & & \mathsf{inv}\, n.\sigma.\alpha.P \;\mid\; \mathsf{accept}\, n \;\mid\; \mathsf{reject}\, n \;\mid\; \mathsf{done}\, n \;\mid\; \mathsf{enough}\, n \\
M & ::= & P \;\mid\; A \;\mid\; M|M
\end{array}$$

We identify terms up to commutativity and associativity of $|$, with $\mathbf{0}$ as identity. This identity is called structural congruence.

The various 'atoms' A are simple those states used in the rendezvous protocol: see Figure 1 for an explanation.

Definition 2 (Transitions) *Messages, ranged over by μ, are*

$$\begin{array}{lcl}
\mu & ::= & \mathsf{off}\, n \;\mid\; \mathsf{reject}\, n \;\mid\; \mathsf{accept}\, n \;\mid\; \mathsf{enough}\, n \\
 & & n = \mathsf{adv}\, \alpha \;\mid\; n = \mathsf{inv}\, n, \alpha \;\mid\; \nu
\end{array}$$

Transitions are as follows. The condition marked $(*)$ is that either $(\alpha = \overline{u}\,\widetilde{x}$, $\beta = u(\widetilde{y})$, $\sigma = \emptyset)$ or $(\alpha = u(\widetilde{y})$, $\beta = \overline{u}\,\widetilde{x}$, $\sigma = \{\widetilde{x}/\widetilde{y}\})$. The broadcast′ rule requires that if M' has a freshly generated name, then it is fresh also with respect to N'.

$$\begin{array}{rcl}
\alpha.P & \xrightarrow{!n=\mathsf{adv}\,\alpha} & \mathsf{adv}\, n.\alpha.P \quad n \textit{ fresh} \\
\mathsf{adv}\, n.\alpha.P & \xrightarrow{!m=\mathsf{adv}\,\alpha} & \mathsf{adv}\, m.\alpha.P \quad m \textit{ fresh} \\
\mathsf{adv}\, n.\alpha.P & \xrightarrow{?m=\mathsf{adv}\,\beta} & \mathsf{hadv}\, m.\sigma.\alpha.P \quad (*) \\
\mathsf{adv}\, n.\alpha.P & \xrightarrow{?m=\mathsf{inv}\,n,\beta} & \mathsf{hinv}\, m.\sigma.\alpha.P \quad (*) \\
\mathsf{hinv}\, n.\sigma.\alpha.P & \xrightarrow{?\,\mathsf{reject}\,n} & \alpha.P \mid \mathsf{enough}\, n \\
\mathsf{hinv}\, n.\sigma.\alpha.P & \xrightarrow{!\,\mathsf{off}\,n} & \mathsf{off}\, n.\sigma.\alpha.P \\
\mathsf{hadv}\, n.\sigma.\alpha.P & \xrightarrow{!m=\mathsf{inv}\,n,\alpha} & \mathsf{inv}\, m.\sigma.\alpha.P \quad m \textit{ fresh} \\
\mathsf{inv}\, n.\sigma.\alpha.P & \xrightarrow{!\,\mathsf{reject}\,n} & \alpha.P \mid \mathsf{reject}\, n \\
\mathsf{inv}\, n.\sigma.\alpha.P & \xrightarrow{?\,\mathsf{off}\,n} & P\sigma \mid \mathsf{accept}\, n \\
\nu x.P & \xrightarrow{!\nu} & P\{x'/x\}\ x' \textit{ fresh} \quad \textit{(new)}
\end{array}$$

$$\frac{M \xrightarrow{!\mu} M' \qquad N \xrightarrow{?\mu} N'}{M|N \xrightarrow{!\mu} M'|N'} \quad \textit{(broadcast′)}$$

$$\begin{array}{rcl}
\mathsf{off}\, n.\sigma.\alpha.P & \xrightarrow{?\,\mathsf{reject}\,n} & \alpha.P \mid \mathsf{enough}\, n \\
\mathsf{off}\, n.\sigma.\alpha.P & \xrightarrow{?\,\mathsf{accept}\,n} & P\sigma \mid \mathsf{enough}\, n \\
\mathsf{accept}\, n & \xrightarrow{?\,\mathsf{enough}\,n} & \mathbf{0} \\
\mathsf{accept}\, n & \xrightarrow{!\,\mathsf{accept}\,n} & \mathsf{accept}\, n \\
\mathsf{reject}\, n & \xrightarrow{?\,\mathsf{enough}\,n} & \mathbf{0} \\
\mathsf{reject}\, n & \xrightarrow{!\,\mathsf{reject}\,n} & \mathsf{reject}\, n \\
\mathsf{done}\, n & \xrightarrow{?\,\mathsf{accept}\,n} & \mathsf{enough}\, n \\
\mathsf{done}\, n & \xrightarrow{?\,\mathsf{reject}\,n} & \mathsf{enough}\, n \\
\mathsf{enough}\, n & \xrightarrow{!\,\mathsf{enough}\,n} & \mathsf{done}\, n \\
M & \xrightarrow{?\mu} & M \quad \textit{(failure)}
\end{array}$$

$$\frac{\alpha.P \xrightarrow{!\mu} S'}{!\alpha.P \xrightarrow{!\mu} !\alpha.P \mid S'} \quad \textit{(replication′)}$$

Write generically $\rightarrow$ to address all transitions $\xrightarrow{!\mu}$.

We have given the *broadcast′* rule in a form familiar from the broadcast calculus of Prasad [24]. However, it should be noted that, up to structural congruence, M is simply a multiset consisting of programs P and atoms A. Note also that all rules apart from *broadcast′* and *replication′* are axioms and operate on single atoms in the multiset. We can remove these last two rules, replacing them with global rules that operate on the entire multiset at once and have flat inference trees:

$$\frac{M_1 \xrightarrow{!\mu} M_1' \quad M_2 \xrightarrow{?\mu} M_2' \ldots M_n \xrightarrow{?\mu} M_n'}{M_1 \mid \ldots \mid M_n \xrightarrow{!\mu} M_1' \mid \ldots M_n'} \qquad \text{(broadcast)}$$

$$\frac{\mu = (!n{=}\,\mathsf{adv}\,\alpha) \quad n \text{ fresh} \quad M_2 \xrightarrow{?\mu} M_2' \ldots M_n \xrightarrow{?\mu} M_n'}{!\alpha.P \mid M_2 \mid \ldots \mid M_n \xrightarrow{!\mu} \mathsf{adv}\, n.\alpha.P \mid !\alpha.P \mid M_2' \ldots M_n'} \qquad \text{(replication)}$$

Clearly, the original and these modified rules both yield the same transition system. Henceforth we use the modified rules. (In both of the rules, any fresh names must be fresh with respect to the entire term.)

Remark 3 (Broadcast and failures) *A characteristic property of a broadcast semantics is that every term is* input-enabled*: ie. $M \xrightarrow{?\mu} M'$ must be defined for every M, μ, even if only to discard it as in $M \xrightarrow{?\mu} M$. Separately, a characteristic of a failure semantics is that messages may be lost: ie. $M \xrightarrow{?\mu} M$. Thus, failures relieve us of the need to separately define every single discard-transition.*

We now define a form of observational equivalence for the machines. We adapt *barbed observation*, familiar from the pi calculus. The idea behind observation is to write $M \downarrow u$ if a third party could observe that machine M can react on u. In our setting, such an observation would be made by hearing the advertisement $!n = \mathsf{adv}\,\overline{u}\,\widetilde{x}$ or $!n = \mathsf{adv}\,u(\widetilde{x})$. The following definition of observation characterises syntactically the states from which such an offer might be broadcast.

Definition 4 (Bisimulation) *The observation relation $M \downarrow u$ is*

$$\begin{array}{rcl} \mathsf{adv}\, n.\overline{u}\,\widetilde{x}.P & \downarrow & u \\ \mathsf{adv}\, n.u(\widetilde{x}).P & \downarrow & u \\ M|N & \downarrow & u \text{ if } M \downarrow u \text{ or } N \downarrow u \end{array}$$

*Given any set equipped with a transition relation $\rightarrow$ and an observation relation $\downarrow$, write $\Rightarrow$ for $\rightarrow^*$ and $\Downarrow$ for $\Rightarrow\downarrow$. **Weak barbed bisimulation** is the largest symmetric relation $\dot{\approx}$ such that, whenever $M \dot{\approx} R$, then (1) $M \downarrow u$ implies $N \Downarrow u$ and (2) $M \rightarrow M'$ implies $N \Rightarrow \dot{\approx} N'$.*

We remark on a slight simplification to the definition of observation. Really, the terms $\overline{u}\,\widetilde{x}.P$ and $!\overline{u}\,\widetilde{x}.P$ can also broadcast advertisements to react on u, and so strictly speaking they should have been added to observation. (Similarly with input). But since these terms are observable already through $\overline{u}\,\widetilde{x}.P \Downarrow u$ and $!\overline{u}\,\widetilde{x}.P \Downarrow u$, there seemed little point.

4 Bisimulation (validity, non-triviality, weak termination)

This article proves three progressively stronger connections between the machines and the pi calculus. The first property is that an interaction for the machine is possible if and only if one is possible for the pi calculus. We actually use the *global pi calculus* [31] (Figure 3), since it admits easier global reasoning – closer to broadcast transactions which are by nature global. Note that pi calculus terms may be considered as multisets of atoms $\alpha.P$, $!\alpha.P$ and $\nu x.P$.

The following translation relates machines M to calculus terms $[\![M]\!]$. Note that the translation is a 'global' translation, in that the meaning of one term off $n.\alpha.P$ depends on whether or not another term $\mathsf{accept}\, n$ is present in the system: if it is present, then we globally know that the interaction n will succeed; if not, then we know that the interaction will fail. The term off $n.\alpha.P$ does not yet locally know the same fact, but it will eventually find out. (Similarly in English law, a contract is deemed to be formed once the acceptance has been posted, even though the offerer does not yet locally know that.)

The **terms** P in the global pi calculus are as for the machine (Definition 1). Terms are identified up to commutativity and associativity of $|$, with $\mathbf{0}$ as identity.

$$\begin{array}{lcl} \alpha & ::= & \overline{u}\,\widetilde{x} \;\mid\; u(\widetilde{x}) \\ P & ::= & \alpha.P \;\mid\; !\alpha.P \;\mid\; \nu x.P \;\mid\; P|P \;\mid\; \mathbf{0} \end{array}$$

The **reaction relation** is the smallest relation $\rightarrow$ satisfying the following axioms:

$$\begin{array}{rcll} \overline{u}\,\widetilde{y}.P \mid u(\widetilde{x}).Q \mid R & \rightarrow & P \mid Q\{\widetilde{y}/\widetilde{x}\} \mid R & \text{(react)} \\ !\overline{u}\,\widetilde{y}.P \mid u(\widetilde{x}).Q \mid R & \rightarrow & P \mid Q\{\widetilde{y}/\widetilde{x}\} \mid !\overline{u}\,\widetilde{y}.P \mid R & \\ \overline{u}\,\widetilde{y}.P \mid !u(\widetilde{x}).Q \mid R & \rightarrow & P \mid Q\{\widetilde{y}/\widetilde{x}\} \mid !u(\widetilde{x}).Q \mid R & \\ !\overline{u}\,\widetilde{y}.P \mid !u(\widetilde{x}).Q \mid R & \rightarrow & P \mid Q\{\widetilde{y}/\widetilde{x}\} \mid !\overline{u}\,\widetilde{y}.P \mid !u(\widetilde{x}).Q \mid R & \\ \nu x.P \mid R & \rightarrow & P\{x'/x\} \mid R \qquad x' \text{ fresh} & \text{(new)} \end{array}$$

Observation $P \downarrow u$ is the smallest relation satisfying

$$\overline{u}\,\widetilde{x}.P \downarrow u \qquad !\overline{u}\,\widetilde{x}.P \downarrow u \qquad u(\widetilde{x}).P \downarrow u \qquad !u(\widetilde{x}).P \downarrow u$$
$$P \mid Q \downarrow u \quad \text{if } P \downarrow u \text{ or } Q \downarrow u$$

Weak bisimulation $\dot{\approx}_\pi$ is as in Definition 4.

Figure 3: The global pi calculus

Definition 5 (Translation) *Define* $\mathsf{accept}(M)$ *to be the set of identifiers* n *such that* $\mathsf{accept}\,n \in M$. *Define* $[\![M]\!] = [\![M]\!]_{\mathsf{accept}(M)}$ *where*

$$[\![P]\!]_o = P$$
$$[\![M|N]\!]_o = [\![M]\!]_o \mid [\![N]\!]_o$$
$$[\![\mathsf{adv}\,n.\alpha.P]\!]_o = [\![\mathsf{hinv}\,n.\sigma.\alpha.P]\!]_o = [\![\mathsf{hadv}\,n.\sigma.\alpha.P]\!]_o = [\![\mathsf{inv}\,n.\sigma.\alpha.P]\!]_o = \alpha.P$$
$$[\![\mathsf{accept}\,n]\!]_o = [\![\mathsf{reject}\,n]\!]_o = [\![\mathsf{done}\,n]\!]_o = [\![\mathsf{enough}\,n]\!]_o = \mathbf{0}$$
$$[\![\mathsf{off}\,n.\sigma.\alpha.P]\!]_o = P\sigma \text{ if } n \in o, \text{ or } \alpha.P \text{ otherwise}$$

From the multiset perspective, $[\![\cdot]\!]$ is a translation from a machine multiset M into a pi calculus multiset P, where each program $Q \in M$ appears in P unchanged and each atom $A \in M$ appears in P as $[\![A]\!]$.

Theorem 6 (Correctness) *Consider bisimulation over the disjoint union of machines and pi calculus terms. Then* $M \dot{\approx} [\![M]\!]$.

The proof is substantial. We start with some explanation. Expanding out the theorem, it amounts to

1. If $M \xrightarrow{M}{}' $ then $[\![M]\!] \Rightarrow_\pi [\![M']\!]$
2. If $[\![M]\!] \rightarrow_\pi P'$ then $M \Rightarrow M'$ such that $[\![M']\!] = P'$
3. If $M \downarrow u$ then $[\![M]\!] \Downarrow u$
4. If $[\![M]\!] \downarrow u$ then $M \Downarrow u$

A corollary is that $P \dot{\approx}_\pi Q$ in the pi calculus if and only if $P \dot{\approx} Q$ in the machine. This corollary is the form of correctness commonly stated for process calculi – it means that the calculus and the machine have the same 'mathematical shape' to their theories – but it is not strong enough for the fairness technique of the following section.

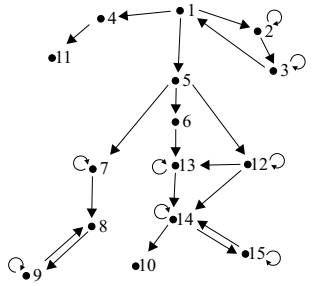
Remark 7 (Analogy) *The traditional semantics of atomic commitment (validity, non-triviality and termination) are weaker forms of bisimulation. The following analogy was suggested by Laura Bocchi. The machine is like a transaction protocol; the pi calculus is like the atomic commitment semantics; and the presence of a sender/receiver pair is like 'yes' votes. Property 1 says that the protocol can only do what the semantics say is possible – ie. validity ('in the presence of 'no' votes then aborting is inevitable'). Property 2 says that the protocol is open to the same possibilities as the high-level semantics – hence non-triviality (in the absence of 'no' votes then commitment is possible) and termination ('one of commitment or aborting is always possible').*

We draw attention to two features of the proof (below) of Theorem 6. The first is **rejectability**. If $[\![M]\!]$ contains $\alpha.P$ then M must have contained $\alpha.P$ either as the program $\alpha.P$, or as translations of an $\mathsf{adv}\, n$, $\mathsf{hinv}\, n$, $\mathsf{hadv}\, n$, $\mathsf{inv}\, n$, $\mathsf{off}\, n$ or $\mathsf{accept}\, n | \mathsf{off}\, n$. And given any of these possibilities, M can abort the transaction they were involved in and start a new one: $M \Rightarrow M'$ such that $[\![M']\!] = [\![M]\!]$ and M' contains the atom $\mathsf{adv}\, m.\alpha.P$ instead of the above.

The second feature is its use of **valid multisets**, as follows. Given a machine M and an identifier n, let $M\big|_n$ be the multiset of atoms within M which include the identifier n. The machine has valid multisets if, for every n, then $M\big|_n$ is one of the following.

1. empty (ie. no states involve n)
2. $\mathsf{adv}\, n.\alpha.P$ and zero or more $\mathsf{hadv}\, n.\sigma_i.\beta_i.P_i$, where each $(\beta_i, \alpha, \sigma_i)$ satisfies (*) from Definition 2.
3. one or more $\mathsf{hadv}\, n.\sigma_i.\beta_i.P_i$
4. $\mathsf{inv}\, n.\sigma.\alpha.P$
5. $\mathsf{inv}\, n.\sigma.\alpha.P$ and $\mathsf{hinv}\, n.\sigma_2.\beta.P_2$ such that (α, β, σ), $(\beta, \alpha, \sigma_2)$ satisfy (*)
6. $\mathsf{inv}\, n.\sigma.\alpha.P$ and $\mathsf{off}\, n.\sigma_2.\beta.P_2$
7. $\mathsf{accept}\, n$ and $\mathsf{off}\, n.\sigma_2.\beta.P_2$
8. $\mathsf{accept}\, n$ and $\mathsf{enough}\, n$
9. $\mathsf{accept}\, n$ and $\mathsf{done}\, n$
10. $\mathsf{done}\, n$
11. $\mathsf{reject}\, n$
12. $\mathsf{reject}\, n$ and $\mathsf{hinv}\, n.\sigma_2.\beta.P_2$
13. $\mathsf{reject}\, n$ and $\mathsf{off}\, n.\sigma_2.\beta.P_2$
14. $\mathsf{reject}\, n$ and $\mathsf{enough}\, n$
15. $\mathsf{reject}\, n$ and $\mathsf{done}\, n$

These valid multisets can rule out anomalous configurations that would never actually arise in practice: for instance, $\mathsf{accept}\, n \mid \mathsf{reject}\, n$. Clearly, a pi calculus program P without atoms is valid. Also, validity is preserved by machine transitions. This diagram shows all possible transitions between valid multisets:



Lemma 8 *If M is well-formed and $M \xrightarrow{!\mu} M'$ then M' is well-formed.*

Proof. A case analysis of the possible transitions. We detail one interesting case. Suppose $M \xrightarrow{!\mu} M'$ was deduced from $\mathsf{hadv}\, n.\sigma.\alpha.P \xrightarrow{!m=\mathsf{inv}\, n,\alpha} \mathsf{inv}\, m.\sigma.\alpha.P$ to give

$$\mathsf{hadv}\, n.\sigma.\alpha.P \mid M_1 \mid \ldots M_n \xrightarrow{!m=\mathsf{inv}\, n,\alpha} \mathsf{inv}\, m.\sigma.\alpha.P \mid M_1' \mid \ldots M_n'$$

for $M_i \xrightarrow{?m=\mathsf{inv}\, n,\alpha} M_i'$ and with m fresh. We first consider $M\big|_n$ and $M'\big|_n$. Given that M is well-formed and includes $\mathsf{hadv}\, n$, then $M\big|_n$ must be multiset 2 or 3. If it is multiset 3, then every $M_i \xrightarrow{?\mu} M_i'$ must be a failure transition $M_i \xrightarrow{?\mu} M_i$, and so $M'\big|_n$ is multiset 3 or 0. If $M\big|_n$ is multiset 2, then one M_i was actually $\mathsf{adv}\, p.\beta.Q$ and it admits either failure or $\xrightarrow{?m=\mathsf{inv}\, n,\alpha} \mathsf{hinv}\, m.\sigma.\beta.Q$. In both cases, the resulting multiset is 2, 3 or 0.

Considering now $M\big|_m$, this must be multiset 0 (since m was generated fresh). The result $M'\big|_m$ must be multiset 5 (if it started with n-multiset 2 and an interaction occurred), or 4 otherwise. □

We now proceed to the main proof.

Proof of Theorem 6 (Correctness) $M \dot{\approx} [\![M]\!]$.

Proof. We make the proof in four parts:

1. If $M \xrightarrow{M'}$ then $[\![M]\!] \Rightarrow_\pi [\![M']\!]$
2. If $[\![M]\!] \rightarrow_\pi P'$ then $M \Rightarrow M'$ such that $[\![M']\!] = P'$
3. If $M \downarrow u$ then $[\![M]\!] \Downarrow u$
4. If $[\![M]\!] \downarrow u$ then $M \Downarrow u$

For Part 1, the transition $M \xrightarrow{!\mu} M'$ was deduced from the broadcast rule (below) or the replication rule (which is similar so we omit it).

$$\frac{M_1 \xrightarrow{!\mu} M_1' \quad M_2 \xrightarrow{?\mu} M_2' \ldots M_n \xrightarrow{?\mu} M_n'}{M_1 \mid \ldots \mid M_n \xrightarrow{!\mu} M_1' \mid \ldots M_n'}$$

In the case where every $[\![M_i]\!] = [\![M_i']\!]$, the result is trivially satisfied. As for the cases which involve some $[\![M_i]\!] \neq [\![M_i']\!]$, we see which transitions they must involve from Definition 5 (translation) and Definition 2 (transitions). And with Lemma 8 (well-formedness) we can enumerate all possible states involving these transitions:

- $\nu x.P \mid M \xrightarrow{!\nu} P\{x'/x\} \mid M$ with x' fresh. The translation $[\![\cdot]\!]$ on both sides gives $\nu x.P \mid [\![M]\!] \rightarrow P\{x'/x\} \mid [\![M]\!]$ with x' fresh – this is just the (new) rule of the global pi calculus.
- $\mathsf{hinv}\, n.\sigma.\alpha.P \mid \mathsf{inv}\, n.\sigma_2.\beta.P_2 \mid M \xrightarrow{!\,\mathsf{off}\, n} \mathsf{off}\, n.\sigma.\alpha.P \mid P_2\sigma_2 \mid \mathsf{accept}\, n \mid M$ with either ($\alpha = \overline{u}\,\widetilde{x}$, $\beta = u(\widetilde{y}), \sigma = \emptyset, \sigma_2 = \{\widetilde{x}/\widetilde{y}\}$) or ($\alpha = u(\widetilde{y}), \beta = \overline{u}\,\widetilde{x}, \sigma = \{\widetilde{x}/\widetilde{y}\}, \sigma_2 = \emptyset$). The translation $[\![\cdot]\!]$ gives $\alpha.P \mid \beta.P_2 \mid [\![M]\!] \rightarrow P\sigma \mid P_2\sigma_2 \mid [\![M]\!]$, which is the (react) rule of the calculus.
- $\mathsf{hinv}\, n.\sigma.\alpha.P \mid \mathsf{inv}\, n.\sigma_2.\beta.P_2 \mid M \xrightarrow{!\,\mathsf{off}\, n} \mathsf{off}\, n.\sigma.\alpha.P \mid \mathsf{inv}\, n.\sigma_2.\beta.P_2 \mid \mathsf{accept}\, n \mid M$. Translation of this and all the following yield the same pi calculus term for both sides, up to structural congruence.
- $\mathsf{hinv}\, n.\sigma.\alpha.P \mid \mathsf{reject}\, n \mid M \xrightarrow{!\,\mathsf{off}\, n} \mathsf{off}\, n.\sigma.\alpha.P \mid \mathsf{reject}\, n \mid M$.
- $\mathsf{inv}\, n.\sigma.\alpha.P \mid \mathsf{off}\, n.\sigma_2.\beta.P_2 \mid M \xrightarrow{!\,\mathsf{reject}\, n} \alpha.P \mid \beta.P_2 \mid \mathsf{enough}\, n \mid M$.
- $\mathsf{reject}\, n \mid \mathsf{off}\, n.\sigma.\alpha.P \mid M \xrightarrow{!\,\mathsf{reject}\, n} \alpha.P \mid \mathsf{enough}\, n \mid M$.

- $\mathsf{accept}\, n \mid \mathsf{off}\, n.\sigma.\alpha.P \mid M \xrightarrow{!\,\mathsf{accept}\, n} \mathsf{accept}\, n \mid P\sigma \mid \mathsf{enough}\, n \mid M.$

For Part 2, The pi calculus step $[\![M]\!] \rightarrow_\pi P'$ can come about in two ways (see Figure 3). Either (1) an element of the translated multiset was $\nu x.P$ and it made a (new) step to become $P\{^{x'}/_x\}$ with the other elements remaining unchanged. Or (2) one element was $\overline{u}\,\widetilde{x}.P$ and another $u(\widetilde{y}).Q$ (or similar replicated variants), and they became P and $Q\{\widetilde{x}/\widetilde{y}\}$ with the other elements unchanged.

If it was a (new) step, then M was either $\nu x.P \mid M_1$, or it was $\mathsf{accept}\, n \mid \mathsf{off}\, n.\alpha.\sigma.(\nu x.P_2) \mid M_1$ such that $(\nu x.P_2)\sigma = \nu x.P$. In the first case, a $!\nu$ transition mirrors the calculus step. In the second, a possible transition

$$\mathsf{accept}\, n \mid \mathsf{off}\, n.\alpha.\sigma.(\nu x.P) \mid M_1 \xrightarrow{!\,\mathsf{accept}\, n} \mathsf{accept}\, n \mid \mathsf{enough}\, n \mid (\nu x.P)\sigma \mid M_1$$

can again be followed by the $!\nu$ transition.

If it was a (react) step, there are more possibilities. The two parties $\overline{u}\,\widetilde{x}.P_1$ and $u(\widetilde{y}).P_2$ (henceforth generically just $\alpha.P$) may each have been present in M already as programs, or as translations of atoms adv, hinv, hadv, inv, or off in the absence or presence of $\mathsf{accept}\, n$. We will show how each possibility can eventually either become the program $\alpha.P$ (which can evolve into $\mathsf{adv}\, n.\alpha.P$) or can become $\mathsf{adv}\, n.\alpha.P$ directly, without altering the translation of the rest of the multiset; similarly $\mathsf{adv}\, m.\beta.P_2$. Then we show how $\mathsf{adv}\, n.\alpha.P$ and $\mathsf{adv}\, m.\beta.P_2$ can react together.

- If $\alpha.P$ was present in $[\![M]\!]$ as $\alpha.P$ or $\mathsf{adv}\, n.\alpha.P$ then the result is already given.
- If present as $\mathsf{hinv}\, m.\sigma.\alpha.P$, then $M\big|_m$ is multiset 5 or 12 (page 4). Multiset 5 gives

  $$\mathsf{hinv}\, m.\sigma.\alpha.P \mid \mathsf{inv}\, m.\sigma_2.\beta.Q \mid N \xrightarrow{!\,\mathsf{reject}\, m} \beta.Q \mid \mathsf{reject}\, m \mid \alpha.P \mid \mathsf{enough}\, m \mid N.$$

  And multiset 12 gives

  $$\mathsf{reject}\, m \mid \mathsf{hinv}\, m.\sigma.\alpha.P \mid N \xrightarrow{!\,\mathsf{reject}\, m} \mathsf{reject}\, m \mid \alpha.P \mid \mathsf{enough}\, m \mid N.$$

- If present as $\mathsf{hadv}\, m.\sigma.\alpha.P$, then $M\big|_m$ is multiset 2 or 3. Multiset 2 gives

  $$\begin{aligned}\mathsf{adv}\, m, \beta.Q \mid \mathsf{hadv}\, m.\sigma.\alpha.P \mid N &\xrightarrow{!p=\mathsf{inv}\, m,\alpha} \mathsf{hinv}\, p.\sigma_2.\beta.Q \mid \mathsf{inv}\, p.\sigma.\alpha.P \mid N \\ &\xrightarrow{!\,\mathsf{reject}\, p} \beta.Q \mid \mathsf{enough}\, p \mid \alpha.P \mid \mathsf{reject}\, p \mid N.\end{aligned}$$

  And multiset 3 gives

  $$\begin{aligned}\mathsf{hadv}\, m.\sigma.\alpha.P \mid N &\xrightarrow{!p=\mathsf{inv}\, m,\alpha} \mathsf{inv}\, p.\sigma.\alpha.P \mid M \\ &\xrightarrow{!\,\mathsf{reject}\, p} \alpha.P \mid \mathsf{reject}\, p \mid N.\end{aligned}$$

- If present as $\mathsf{inv}\, m.\sigma.\alpha.P$, then $M\big|_m$ is multiset 4, 5 or 6. Multiset 4 gives

  $$\mathsf{inv}\, m.\sigma.\alpha.P \mid N \xrightarrow{!\,\mathsf{reject}\, m} \alpha.P \mid \mathsf{reject}\, m \mid M.$$

  And multiset 5 gives

  $$\mathsf{inv}\, m.\sigma.\alpha.P \mid \mathsf{hinv}\, m.\sigma_2.\beta.Q \mid N \xrightarrow{!\,\mathsf{reject}\, m} \alpha.P \mid \mathsf{reject}\, m \mid \beta.Q \mid \mathsf{enough}\, m \mid N.$$

  And multiset 6 gives

  $$\mathsf{inv}\, m.\sigma.\alpha.P \mid \mathsf{off}\, m.\sigma_2.\beta.Q \xrightarrow{!\,\mathsf{reject}\, m} \alpha.P \mid \mathsf{reject}\, m \mid \beta.Q \mid \mathsf{enough}\, m \mid N.$$

- If present as $\mathsf{off}\ m.\sigma.\alpha.P$ without a corresponding $\mathsf{accept}\ m$ then $M\big|_m$ is multiset 6 or 13. Multiset 6 gives

$$\mathsf{inv}\ m.\sigma_2.\beta.Q \mid \mathsf{off}\ m.\sigma.\alpha.P \quad \stackrel{!\,\mathsf{reject}\ m}{\longrightarrow} \quad \beta.Q \mid \mathsf{reject}\ m \mid \alpha.P \mid \mathsf{enough}\ m \mid N.$$

And multiset 13 gives

$$\mathsf{reject}\ m \mid \mathsf{off}\ m.\sigma.\alpha.P \mid N \quad \stackrel{!\,\mathsf{reject}\ m}{\longrightarrow} \quad \mathsf{reject}\ m \mid \alpha.P \mid \mathsf{enough}\ m \mid N.$$

- If $\alpha.P$ was present in $[\![M]\!]$ as $\mathsf{off}\ m.\sigma.\beta.Q \mid \mathsf{accept}\ m$ such that $Q\sigma = \alpha.P$ then it must have been multiset 7. This gives

$$\mathsf{accept}\ m \mid \mathsf{off}\ m.\sigma.\beta.Q \mid N \quad \stackrel{!\,\mathsf{accept}\ m}{\longrightarrow} \quad \mathsf{accept}\ m \mid Q\sigma \mid \mathsf{enough}\ m \mid N.$$

Finally, we have a machine which contains $\mathsf{adv}\ n.\overline{u}\,\widetilde{x}.P \mid \mathsf{adv}\ m.u(\widetilde{y}).Q$. We show how they react to yield $P \mid Q\{\widetilde{x}/\widetilde{y}\}$, thus matching the calculus.

$$\begin{aligned}
&\mathsf{adv}\ n.\overline{u}\,\widetilde{x}.P \mid \mathsf{adv}\ m.u(\widetilde{y}).Q \mid N \\
\stackrel{!n'=\mathsf{adv}\ \overline{u}\,\widetilde{x}}{\longrightarrow}\ &\mathsf{adv}\ n'.\overline{u}\,\widetilde{x}.P \mid \mathsf{hadv}\ n'.\{\widetilde{x}/\widetilde{y}\}.u(\widetilde{y}).Q \mid N \\
\stackrel{!m'=\mathsf{inv}\ n',u(\widetilde{y})}{\longrightarrow}\ &\mathsf{hinv}\ m'.\{\,\}.\overline{u}\,\widetilde{x}.P \mid \mathsf{inv}\ m'.\{\widetilde{x}/\widetilde{y}\}.u(\widetilde{y}).Q \mid N \\
\stackrel{!\,\mathsf{off}\ m'}{\longrightarrow}\ &\mathsf{off}\ m'.\{\,\}.\overline{u}\,\widetilde{x}.PQ\{\widetilde{x}/\widetilde{y}\} \mid \mathsf{accept}\ m' \mid N \\
\stackrel{!\,\mathsf{accept}\ m'}{\longrightarrow}\ &P \mid Q\{\widetilde{x}/\widetilde{y}\} \mid \mathsf{enough}\ m' \mid \mathsf{accept}\ m' \mid N.
\end{aligned}$$

For Part 3, if $M \downarrow u$, then the multiset M contains either $\mathsf{adv}\ n.\overline{u}\,\widetilde{x}.P$ or $\mathsf{adv}\ n.u(\widetilde{x}).P$. Therefore the translated multiset $[\![M]\!]$ contains $\overline{u}\,\widetilde{x}.P$ or $u(\widetilde{x}).P$ and also has an observation on u.

For Part 4, suppose $[\![M]\!] \downarrow u$. This must be because the translated multiset contains $\overline{u}\,\widetilde{x}.P$ or $u(\widetilde{x}).P$. Which in turn is because the original multiset M contained the same $\alpha.P$ either as $\alpha.P$ directly, or as the translation of atoms adv, hinv, hadv, inv, or off. As in Part 2, M does transitions to transform this atom into $\mathsf{adv}\ n.\alpha.P$ without altering the translation of the rest of the machine. And $\mathsf{adv}\ n.\alpha.P$ yields the desired barb. □

5 Fairness (weakly non-blocking)

TODO: Cite Prakash.

In this section we assume fair communication failures with no site failures, and we prove *weak non-blocking* in the machine: if the calculus admits some transitions, then the corresponding machine will 'eventually' perform one of the transitions. Fairness holds on a millisecond scale for packet loss in wireless networks (Observation 11), and for packet loss due to Internet congestion [30]. On a scale of minutes, it holds for web services which are soon rebooted when they crash.

The technique introduced in this section is quite general: it applies to any calculus, and any implementation of that calculus in a failure setting, so long as the two are bisimilar. The technique is to take the machine transition system, 'cross' it with a failure transition system, and use a standard lemma about the probability of eventually exiting a finite transition system. To construct the 'cross' properly we have to avoid certain pathological cases:

1. Assume that the machine has 'reasonable' time intervals between each step it takes: it does not perform an infinite number of steps before communication failure has a chance to be repaired, nor does it stall forever.

2. Assume that failure transitions are independent of current machine state. This disallows a 'failure demon' who makes communication fail whenever the machine looks like making useful progress, and restores communication otherwise.

3. Assume that machine transitions are independent of the failure process. For instance, this disallows the programmer from knowing that failures always happen on every even hour of the day, and scheduling machine transitions to only take place on even hours.

Definition 9 (Machine and calculus) *A distributed machine is a finite set of distributed locations $\ell \in \mathcal{L}$, a set of states $M \in \mathcal{M}$ and a transition relation $M \xrightarrow{\ell_1,\dots} M' \in \mathcal{M} * \wp(\mathcal{L}) * \mathcal{M}$. Write $\to$ to denote any labelled transition, and $\Rightarrow$ for $\to^*$*

A calculus is a set of states $P \in \mathcal{P}$ and a translation relation $P \to P' \in \mathcal{P} * \mathcal{P}$.

A correctness-preserving translation $[\![\cdot]\!] : \mathcal{M} \mapsto \mathcal{P}$ is one where (1) $M \to M'$ implies $[\![M]\!] \Rightarrow [\![M']\!]$ and also (2) $[\![M]\!] \to P'$ implies $M \Rightarrow M'$ such that $[\![M']\!] = P$. A finite translation is one were the set $\{M : [\![M]\!] = P\}$ is finite for all P. Denote this set $\mathcal{M}\big|_P$.

The annotation on the transition relation $\xrightarrow{\ell_1,\dots}$ indicates that this transition requires a successful network message to be sent between the listed locations $\ell_1, \dots$. We remark that correctness $M \mathrel{\dot\approx} [\![M]\!]$ (Theorem 6) implies that $[\![\cdot]\!]$ is correctness-preserving as defined above; the more traditional correctness statement $M \mathrel{\dot\approx} N \iff [\![M]\!] \mathrel{\dot\approx} [\![N]\!]$ does not.

Definition 10 (Failures) *A fair failure model is a Markov process $(F_t,\ t \geq 0)$ with finite state space $\mathcal{F}$, with a function* okay *$: \mathcal{F} \mapsto \wp(\mathcal{L} \times \mathcal{L})$.*

The okay function denotes that, in a failure-state F with $(\ell_1, \ell_2) \in \mathsf{okay}(F)$, then the network connection between ℓ_1 and ℓ_2 is 'up' and so a message between them will succeed.

Observation 11 (Wireless) *Wireless has a fair failure model.*

Proof. The standard model for failures in a wireless network is known as the Gilbert model [16]. This says that the link quality between any two parties evolves as a Markov process where in each state the bit errors are independent. On practical grounds, Lemmon reports that the model is empirically very accurate for a describing a range of wireless networks and land mobile radio [17]. In practice, the probability of an individual message being lost ranges from 0.001 to 0.15 [29]. Write $\mathcal{F}_{\ell_1,\ell_2}$ for the quality between ℓ_1 and ℓ_2, with characteristic function okay $: \mathcal{F}_{\ell_1,\ell_2} \mapsto$ *boolean*. Since there are finitely many locations, then the pairwise product between all locations straightforwardly induces a fair failure model. □

We now embed the machine's transitions into the failure process, to make an *embedded Markov chain*. The following definition states formally the three assumptions at the start of this section:

Assumption 12 (Embedded Markov chain) *Let M_0 be the initial state of the machine, in which it is at time $T_0 = 0$. Let M_n be the state of the machine after n transitions and T_n be the time at which it makes the nth transition. Call the pair (M_n, F_{T_n}) a configuration.*

1. *Assume (M_n, F_{T_n}) is a Markov chain and that, conditional on (M_n, F_{T_n}), $(F_t : t \geq T_n)$ is independent of M_n.*
2. *Assume a collection D_M of random variables and suppose that none of them is trivial – ie. for each $M \in \mathcal{M}$ there exist $0 < a_M < b_M < \infty$ such that $\mathbb{P}(a < M < D_M < b_M) > 0$. Conditional on $M_n = M$, let $T_{n+1} - T_n$ be distributed like D_M and let it be independent of the failure process.*
3. *Assume a collection $J_{M,F}$ of random variables. Conditional on $(M_n, F_{T_{n+1}})$, let M_{n+1} be distributed like $J_{M_n,F_{T_{n+1}}}$ and let it be independent of the failure process and of $T_{n_1} - T_n$.*

TODO: Is indep. of failure already implied?

The key result is that if a transition is possible in the right failure state, then it is possible from any failure state:

Proposition 13 (Possibility) *Assume an embedding, as per above. Given a machine transition $M \to M'$, suppose there is a failure state F such that $\mathbb{P}(J_{M,F} = M') > 0$. Then for all failure states G,*

$$\mathbb{P}(M_{n+1} = M' \mid M_n{=}M,\ F_{T_n}{=}G) > 0.$$

Proof. The idea is that the system starts in configuration (M_n, G), and is going to do a machine transition at a given time T_{n+1}. We establish that the failure state can change from G to f before that time, and can stay at F until after that time. By the assumption, within that time period, the desired transition is possible.

By Assumption 12.2 there is a time interval $[a, b]$ such that $T_n < a < b$ and $\mathbb{P}(a < T_{n+1} < b) > 0$. Then

$$\begin{aligned}&\mathbb{P}(M_{n+1}{=}M' \mid M_n{=}M,\ F_{T_n}{=}g)\\ \geq\ &\mathbb{P}(a{<}T_{n+1}{<}b,\ M_{n+1}{=}M',\ F_t{=}f \text{ for all } t \in [a,b] \ \mid\ M_n{=}M,\ F_{T_n}{=}g)\end{aligned}$$

since the latter refers to a smaller event. Now factor out the conditions, using the conditional probability theorem – that $\mathbb{P}(A, B \mid C) = \mathbb{P}(A, B, C) = \mathbb{P}(A \mid B, C).\mathbb{P}(B \mid C)$:

$$= \mathbb{P}(M_{n+1}{=}M' \ \mid\ M_n{=}M,\ F_{T_n}{=}G,\ F_t{=}F \text{ for all } t \in [a,b], a{<}T_{n+1}{<}b). \tag{1}$$

$$\mathbb{P}(a{<}T_{n+1}{<}b \ \mid\ M_n{=}M,\ F_{T_n}{=}G,\ F_t{=}F \text{ for all } t \in [a,b]). \tag{2}$$

$$\mathbb{P}(F_t{=}F \text{ for all } t \in [a,b] \ \mid\ M_n{=}M,\ F_{T_n}{=}G) \tag{3}$$

By Assumption 12.1, future failures are independent of current machine state. This allows us to rewrite:

$$(3) = \mathbb{P}(F_t{=}F \text{ for all } t \in [a,b] \ \mid\ F_{T_n}{=}G).$$

Since (F_t) is a Markov process (Definition 10) then $(3) > 0$.

By Assumption 12.2, transition times are independent of failure states. This allows us to rewrite:

$$(2) = \mathbb{P}(a{<}T_{n+1}{<}b \ \mid\ M_n{=}M).$$

By choice of a and b, we get $(2) > 0$.

By Assumption 12.2, machine transitions are independent of time intervals and the failure process. This allows us to rewrite:

$$(1) = \mathbb{P}(M_{n+1}{=}M' \ \mid\ M_n{=}M,\ F_{T_{n+1}}{=}F).$$

By the assumption of the proposition, $(1) > 0$.

Hence $(1).(2).(3) > 0$, as required. □

Proposition 14 (Progress) *Given a fair failure model, if a transition $[\![M]\!] \to P'$ is possible in the calculus, then it is overwhelmingly likely that eventually the machine will make a transition $M \Rightarrow M'$ such that the calculus matches it $[\![M]\!] \to [\![M']\!]$.*

Proof. Consider the Markov chain (M_n, F_{T_n}). Project this onto the set of states $M_n \in \mathcal{M}\big|_P \cup \{\textit{outside}\}$, with all $\{(M_i, F_{T_i}) : \exists j \leq i. M_j \notin \mathcal{M}\big|_P\}$ being projected onto $(\textit{outside}, F_{T_i})$. The result is still a Markov chain. Note that *outside* includes all states P' such that $[\![M]\!] \to P'$.

By assumption there exists a series of machine transitions $M \to M_1 \ldots \to M_n \to \textit{outside}$. By Proposition 13, for any F, there exists a path in the Markov chain $(M, F) \to (M_1, F_1) \ldots \to (M_n, F_n) \to (\textit{outside}, ?)$ with positive probability. By the Borel-Cantelli lemma, (M, F) is transient for all F (ie. is visited only finitely often). □

Applicability

The corollary above applies immediately to a real implementation, where $\mathcal{M}\big|_P$ is always finite. But in the formalism we have introduced, it is not finite. For instance, $\mathcal{M}\big|_{\mathbf{0}}$ includes all subsets of the infinite set $\{\mathsf{okay}\, n : n \text{ is an identifier}\}$.

Instead, quotient the set of machines up to the presence or absence of all $\mathsf{accept}\, n$, $\mathsf{reject}\, n$, $\mathsf{done}\, n$ and $\mathsf{enough}\, n$ such that $M\big|_n$ contains no mentions of n other than these four atoms. This quotiented machine then satisfies the finiteness requirement, and has exactly the same behaviour as the original.

6 Types (strongly non-blocking)

In this section we assume the possibility of site failure and of communication failure. Actually we treat site failure merely as 'unfair' communication failure – i.e. all messages to and from a site are lost, forever.

In the protocol, so long as the inviting party has not crashed, it can always spontaneously reject the current rendezvous. Thus, without needing to wait for a reply, it becomes free for subsequent rendezvous. In particular,

$$\mathsf{hadv}\, n.\sigma.\alpha.P \quad \stackrel{!m=\mathsf{inv}}{\longrightarrow} \quad \mathsf{inv}\, m.\sigma.\alpha.P \quad \stackrel{!\,\mathsf{reject}\, m}{\longrightarrow} \quad \alpha.P. \tag{4}$$

From the transaction perspective this result is not surprising – the inviting party acts as a coordinator who can authoritatively decide to abort. A practical application is to make a web service always be the inviting party. This way the web service will never block in any situation. (The client might block, in the particular situation where the client makes advertises and the service invites offers but then crashes before accepting the offer. We feel this is reasonable, since web services are normally restored within minutes of a crash.)

One is tempted to say that the sender should always advertise, and the receiver should always invite. However this is not useful. Data only ever flows from sender to receiver, so a receive-only web service could never emit data. And vice versa, if receivers always advertise and senders always invite, then the web service could never take in requests. This explains why we designed the protocol to allow offers from both senders and receivers.

In this section we modify the transition system to separate advertisers from inviters, through use of *polarities*. A positive name u^+ will advertise when it executes 'send' commands, and a negative name v^- will advertise when it executes 'receive' commands. We then give a type system: if a program is *invite-typed*, then during its execution it will only ever play the role of inviter. The type system is Milner's *sorting* [19], coupled with polarities.

For modifying the transition system, we remove the advertisement transitions from Definition 2:

$$\alpha.P \quad \stackrel{!n=\mathsf{adv}\,\alpha}{\longrightarrow} \quad \mathsf{adv}\, n.\alpha.P \qquad n \textit{ fresh} \tag{advertise}$$

$$\mathsf{adv}\, n.\alpha.P \quad \stackrel{?m=\mathsf{adv}\,\beta}{\longrightarrow} \quad \mathsf{hadv}\, m.\sigma.\alpha.P \tag{hear}$$

We replace them with the following polarity-respecting transitions:

$$\overline{u}^{+}\widetilde{x}.P \stackrel{!n=\mathsf{adv}\,\overline{u}^{+}\widetilde{x}}{\longrightarrow} \mathsf{adv}\, n.\overline{u}^{+}\widetilde{x}.P \qquad n \textit{ fresh} \tag{advertise+}$$

$$u^{-}(\widetilde{x}).P \stackrel{!n=\mathsf{adv}\,u^{-}(\widetilde{x})}{\longrightarrow} \mathsf{adv}\, n.u^{-}(\widetilde{x}).P \qquad n \textit{ fresh} \tag{advertise–}$$

$$u^{+}(\widetilde{x}).P \stackrel{?n=\mathsf{adv}\,\overline{u}^{+}\widetilde{y}}{\longrightarrow} \mathsf{hadv}\, n.\{\widetilde{y}/\widetilde{x}\}.u^{+}(\widetilde{x}).P \tag{hear+}$$

$$\overline{u}^{-}\widetilde{x}.P \stackrel{?n=\mathsf{adv}\,u^{-}(\widetilde{y})}{\longrightarrow} \mathsf{hadv}\, n.\{\,\}.\overline{u}^{-}\widetilde{x}.P \tag{hear–}$$

Definition 15 (Polarities) *Partition names into two sets: the positive names $u^+, \ldots$ and the negative names $v^-, \ldots$. Let undecorated names x range over both positive and negative names. Assume a set $\mathcal{S}$ of sorts ranged over by $T, U \ldots$, again partitioned into positive and negative, with a function* $\mathrm{sort} : \mathcal{S} \mapsto \mathcal{S}^*$.

A sorting is an assignment of sorts to the names in a machine such that, within a name's scope, every occurrence of it has the same sort. Write $x : T$ for when x has sort T. A well-sorting is one where

1. $\mathrm{sort}(T_0) = T_1 \ldots T_n$ *holds for every receive command $x_0{:}T_0(x_1{:}T_1 \ldots x_n{:}T_n).P$ and every send command $\overline{x}_0{:}T_0\ x_1{:}T_1 \ldots x_n{:}T_n.P$.*

2. *Also, for each i, the polarity of x_i matches that of T_i.*

3. *In* $\mathsf{hinv}\, n.\sigma.\alpha.P$, $\mathsf{off}\, n.\sigma.\alpha.P$, $\mathsf{hadv}\, n.\sigma.\alpha.P$ *and* $\mathsf{inv}\, n.\sigma.alpha.P$, *the substitution σ only ever substitutes a name with another of the same sort.*

A machine is invite-typed if it contains no $\overline{u}^{+}\widetilde{x}.P$ or $u^{-}(\widetilde{x}).P$, and no $\mathsf{adv}\, n.\alpha.P$, $\mathsf{hinv}\, n.\sigma.\alpha.P$ *or* $\mathsf{off}\, n.\sigma.\alpha.P$.

Proposition 16 (Subject reduction) *All machine transitions preserve well-sortedness. Moreover, given well-sortedness, they also preserve invite-typing.*

Proof. Consider first well-sortedness. Most transitions have the same $\alpha.P$ and σ in their left and right, and so trivially preserve well-sortedness. The other transitions are as follows.

1. The *hear–* transition introduces an empty substitution, which is trivially well-sorted.
2. The *hear+* transition introduces involves $u^+(\widetilde{x}).P$ and $\overline{u}^+\widetilde{y}$ and introduces the substitution $\sigma = \{\widetilde{y}/\widetilde{x}\}$. But $\widetilde{x}$ and $\widetilde{y}$ have the same sorts, and so σ preserves sorts.
3. The other transitions $\mathsf{inv}\, n.\sigma.\alpha.P \xrightarrow{?\,\mathsf{off}\, n} P\sigma \mid \mathsf{accept}\, n$ and $\mathsf{off}\, n.\sigma.\alpha.P \xrightarrow{?\,\mathsf{accept}\, n} P\sigma \mid \mathsf{enough}\, n$. These both apply a well-sorted substitution to a term – such substitution preserves well-sortedness.

As for preservation of invite-typing, suppose $M \to M'$ with M invite-typed and well-sorted. We wish to prove the invite-type properties hold for M'. We do this by considering the possible transitions (Definition 2, and the revised *advertise* and *hear* transitions above).

1. To prove that M' has no $\overline{u}^+\widetilde{x}$. All transitions have identical $\alpha.P$ on left and right, apart from those that apply a substitution σ to turn $\overline{u}\,\widetilde{x}$ into $\overline{u}\sigma\ \widetilde{x}\sigma$. But σ is always well-sorted, and so $u\sigma$ has the same polarity as u. Therefore M' has $\overline{v}^+\widetilde{y}$ if and only if M has some $\overline{u}^+\widetilde{x}$. The input case is similar.
2. To prove that M' has no $\mathsf{adv}\, n.\alpha.P$. The only transitions that yield adv on the right hand side are the *advertise* transitions (which can never apply since M has no $\overline{u}^+\widetilde{x}$ or $u^-(\widetilde{y})$); or the transition $\mathsf{adv}\, n.alpha.P \xrightarrow{!m=\mathsf{adv}\,\alpha} \mathsf{adv}\, m.\alpha.P$ (which can never apply since M has no adv).
3. To prove that M' has no $\mathsf{hinv}\, n.\alpha.P$. The only transition with hinv on the right hand side is $\mathsf{adv}\, n.\alpha.P \xrightarrow{?m=\mathsf{inv}\, n,\beta} \mathsf{hinv}\, m.\sigma.\alpha.P$. This can never apply, since M has no adv. A similar argument shows that M' has no $\mathsf{off}\, n.\sigma.\alpha.P$, which can only have come from inv. □

We now proceed to the non-blocking result. The situation is that the system starts with two parts, a invite-typed web server program M and a client N. We keep the two separate with a comma M, N, which behaves like | but is non-commutative. Formally, this involves additional broadcast rules. As usual, any names freshly generated by one part must be globally fresh.

$$\frac{M \xrightarrow{!\mu} M' \qquad N_1 \xrightarrow{?\mu} N_1' \ldots N_n \xrightarrow{?\mu} N_n'}{M, N_1| \ldots |N_n \xrightarrow{!\mu} M', N_1'| \ldots |N_n'}$$

$$\frac{M_1 \xrightarrow{?\mu} M_1' \ldots M_m \xrightarrow{?\mu} M_m' \qquad N \xrightarrow{!\mu} N'}{M_1| \ldots |M_1', N \xrightarrow{!\mu} M_1'| \ldots |M_n', N'}$$

The situation starts with the web server and the client:

$$M, N$$

The two may perform several interactions $\Rightarrow$. Note that any reduction $M, N \to M', N'$ preserves well-sortedness and invite-typing of each part individually.

$$M', N'$$

Suddenly, N' fails:

$$M', \mathbf{0}$$

At this point, M' might have been part way through the rendezvous protocol with N', and so it might have atoms in any of the intermediate states $\mathsf{hadv}\, n.\sigma.\alpha.P$ or $\mathsf{inv}\, n.\sigma.\alpha.P$ – generically

we just write $\alpha.P \in [\![M]\!]$. (Since M' is invite-typed, it never has $\mathsf{adv}\, n.\alpha.P$ or $\mathsf{hinv}\, n.\sigma.\alpha.P$ or $\mathsf{off}\, n.\sigma.\alpha.P$).

The following proposition is that M' can still react. It is phrased somewhat carefully. *If* M' has a send or receive command in it (even one that had been part way through a protocol with N'), and *if* someone then adds a complimentary receive or send command into the system, then the two can rendezvous *together*. Complementarity comes from (*) in Definition 2. In the proposition, togetherness is implied by the fact that the complementary command has been used up. (The only way for it to have been used up is through reaction with M'.)

Proposition 17 (Non-blocking) *Let M' be invite-typed and $\alpha.P \in [\![M]\!]$. If α and some β, σ satisfy (*) from Definition 2, then $M', \beta.\mathbf{0} \Rightarrow M'', \mathsf{done}\, n$ for some M'', n.*

Proof. Because of invite-typing, α is either $u^+(\widetilde{x})$ or $\overline{u}^-\widetilde{x}$. We consider the first case, where β is $\overline{u}^+\widetilde{y}$. The second case is similar.

From the invite-typing of M' and the translation $[\![\cdot]\!]$ (Definition 5), then M' contains either $u^+(\widetilde{x}).P$ directly, or one of the atoms $\mathsf{hadv}\, n.\sigma.u^+(\widetilde{x}).P$ or $\mathsf{inv}\, n.\sigma.u^+(\widetilde{x}).P$. From Equation 4 (page 17) both of these atoms reduce to $u^+(\widetilde{x}).P$ (similar to *rejectability* in Section 4).

Given M' either containing $u^+(\widetilde{x}).P$ or reducing to it, we continue reaction in the presence of $\beta.\mathbf{0}$. (The first transition below involves the polarised rules *advertise+* on the part of $\beta.\mathbf{0}$, and *hear+* on the part of M'.)

$$
\begin{aligned}
& u^+(\widetilde{x}).P \mid M_1, \quad \overline{u}^+\widetilde{y}.\mathbf{0} \\
\xrightarrow{!m=\mathsf{adv}\,\overline{u}^+\widetilde{x}} \quad & \mathsf{hadv}\, m.\{\widetilde{y}/\widetilde{x}\}.u^+(\widetilde{x}).P \mid M_1, \quad \mathsf{adv}\, m.\overline{u}^+\widetilde{y}.\mathbf{0} \\
\xrightarrow{!n=\mathsf{inv}\, m, u^+(\widetilde{x})} \quad & \mathsf{inv}\, n.\{\widetilde{y}/\widetilde{x}\}.u^+(\widetilde{x}).P \mid M_1, \quad \mathsf{hinv}\, n.\{\,\}.\overline{u}^+\widetilde{y}.\mathbf{0} \\
\xrightarrow{!\,\mathsf{off}\, n} \quad & P\{\widetilde{y}/\widetilde{x}\} \mid \mathsf{accept}\, n \mid M_1, \quad \mathsf{off}\, n.\{\,\}.\overline{u}^+\widetilde{y}.\mathbf{0} \\
\xrightarrow{!\,\mathsf{accept}\, n} \quad & P\{\widetilde{y}/\widetilde{x}\} \mid \mathsf{accept}\, n \mid M_1, \quad \mathsf{enough}\, n \\
\xrightarrow{!\,\mathsf{enough}\, n} \quad & P\{\widetilde{y}/\widetilde{x}\} \mid M_1, \quad \mathsf{done}\, n
\end{aligned}
$$

□

7 Conclusions

In the process calculi literature, one often reads 'we use asynchronous rendezvous, because it is more suited than synchronous to a distributed network.' This statement is only a half truth. More precisely, asynchronous is well suited to *modelling* a distributed network at low level, but in the presence of failures it is awful for *programming* it. Some kind of reliable transport layer is always needed, either to guarantee that the message has arrived or to guarantee that it has performed its synchronous rendezvous. This paper gives a reliable implementation of synchronous rendezvous on top of an asynchronous fallible network. This can be used either as the basis for a programming language, or for modelling the network at a slightly higher level.

We consider how the solution applies to some standard problems. Recall the Two Generals problem from Section 1: two generals have only unreliable communication between themselves, but they must either both decide to attack or both decide not to. Equivalently, the program $\overline{u}.P \mid u().Q$ must either unblock both P and Q or it must not react. The problem is often said to be insoluble, by the following reasoning. Consider the shortest sequence of messages which guarantees that both generals will attack. The protocol must also work in the presence of failure, and so it does not depend on the successful transmission of the final message. Therefore a simpler protocol exists which never even uses that final message – a contradiction. But this reasoning only proves that there exists no bounded protocol. The protocol in this paper is unbounded. However from Section 5 we know that the probability of the protocol not terminating tends to zero as time increases. In effect, there is an 'eventual' solution to the two generals problem.

The standard problem of atomic commitment is often said to have no solution in the presence of arbitrary message loss as well as site failure. The result is actually more precise [27]: at least

1. Romeo advertises to all the pretty ladies in Verona, *anyone up for a date?*
2. Many (including Juliet) invite him to make offer: *what do you have in mind?*
3. Romeo offers to Juliet, *dinner!* Her decision will now be final.
4. Juliet replies *I accept dinner!* Now Romeo also knows the conclusion.
5. Romeo replies *enough already with the accepting!*

Figure 4: An instance of our three phase commit protocol, as used in Verona. Juliet could have changed her mind at any time between messages 2 and 4. Romeo cannot make two different offers at the same time. No one dies.

some committing nodes will be forced to block until they can re-establish communication with the other nodes. This result is a serious problem for distributed transactions, where there are potentially many committing nodes. But in the special case of rendezvous there are only two committing nodes – the sender and the receiver. Moreover from Section 6 we know that only the client and never the web server will be forced to block. This is not so serious, since a web service will normally be restored promptly in the event of failure.

The same limitation has been stated [12] in terms of a 'window of vulnerability' – an interval during which the failure of one node can cause the entire protocol to wait indefinitely. In our protocol, the wait is not caused by the failure of *any* one node but only by the failure of one *particular* node (the server). And in this case, the *'entire protocol'* that waits is composed just of the client. (The vulnerable window exists solely in the client; it starts when the client receives an acceptance and ends when it receives *okay* or *abort*).

The final standard problem is that of star-cross'd lovers who arrange secret rendezvous. As we see from the example of Romeo and Juliet [25] (who committed suicide after a rendezvous failure due to message loss), reliable rendezvous is literally matter of life or death. The solution is to adopt the three phase commit protocol discussed in this paper. An instance of this 'Verona protocol' is sketched in Figure 4. The full version below is due Vivika McKie-Woods; numbers in parentheses refer to the figure. Because of literary constraints it has been rendered as a poem rather than a tragedy. We remark that message-loss is as relevant today with SMS messages and emails as it was in the era of Romeo and Juliet.

Romeo was lonely so he messaged 'Seeking Date. (1)
Who fancies meeting later? Around eightish – don't be late!'
Juliet was wary (she'd been warned about his kind)
'I won't accept unless you tell me what you have in mind.' (2)
'I offer dinner, candles and then after we can dance.' (3)
It sounded harmless – Julie thought she'd give the boy a chance.
'I have a car, I'll pick you up' he said, or rather sleazed.
'Best not', she answered, 'for my parents would not be best pleased.
Meet me by the fountain. We can savour the night air.' (4)
'Victory at last! I mean, Thats great, I'll see you there!' (5)
Our saga ended merrily for each had used their head,
They enjoyed a lovely night – and no one wound up dead!

References

[1] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.

[2] BEA, IBM, Microsoft, SAP, and Siebel. Business process execution language for web services (BPEL4WS). http:// ifr.sap.com/ bpel4ws/.

[3] M. Berger and K. Honda. The two-phase commitment protocol in an extended pi-calculus. In *EXPRESS '00*, volume 39 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers.

[4] P. Bernstein. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman, 1987.

[5] F. Bisi, S. Carpineti, C. Laneve, P. Milazzo, and L. Wischik. The Bologna pi language, compiler and runtime. In progress.

[6] L. Bocchi. Compositional nested transactions. Submitted for publication.

[7] L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long running transactions. In *FMOODS 2003*. To appear.

[8] R. Bruni, C. Laneve, and U. Montanari. Orchestrating transactions in join calculus. In *CONCUR 2002*, LNCS 2421:321–337.

[9] Business Process Management Initiative. Business process modelling notation (BPML). http:// www.bpmi.org/.

[10] S. Conchon and F. L. Fessant. Jocaml: Mobile agents for objective-caml. In *ASA/MA '99*.

[11] M. Corporation. Biztalk server. http:// www.microsoft.com/ biztalk/.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[13] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of POPL '96*, pages 372–385. ACM Press.

[14] P. Gardner, C. Laneve, and L. Wischik. Linear forwarders. In *CONCUR 2003*, LNCS 2761:415–430.

[15] A. Giacalone, P. Mishra, and S. Prasad. FACILE: A symmetric integration of concurrent and functional programming. *International Journal of Parallel Programming*, 18(2):121–160, 1989.

[16] E. N. Gilbert. Capacity of a burst-noise channel. *The Bell System Technical Journal*, 39:1253–1265, 1960.

[17] J. J. Lemmon. Wireless link statistical bit error model. *US National Telecommunications and Information Administration Report*, 02-394, 2002.

[18] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1997.

[19] R. Milner. The polyadic pi calculus: A tutorial. In *Logic and Algebra of Specification, NATO ASI 1991*, volume 94 of *Series F*. Springer-Verlag.

[20] R. Milner. *Communicating and mobile systems: the Pi-calculus*. Cambridge University Press, 1999.

[21] U. Nestmann, R. Fuzzati, and M. Merro. Modeling consensus in process calculus. In *CONCUR 2003*, LNCS 2761:400–414.

[22] Oasis Business Transaction Protocol Committee. Business transaction protocol 1.0 specification. http:// www.oasis-open.org/ committees/ download.php/ 1184/ 2002-06-03.BTP_cttee_spec_1.0.pdf, 2003.

[23] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Foundations of Computing, pages 455–494. MIT Press, 2000.

[24] K. V. S. Prasad. A calculus of broadcasting systems. In *TAPSOFT 1991*, LNCS 493:338–358.

[25] W. Shakespeare. Romeo and Juliet. Manuscript, 1599.

[26] D. Skeen. A quorum-based commit protocol. In *6th Berkely Workshop on Distributed Data Management And Computer Networks*, pages 69–80, 1982. Also as Technical Report TR82-483, Computer Science Dept, Cornell University.

[27] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, SE-9:219–228, 1983.

[28] S. Thatte. XLANG: Web services for business process design. http://www.gotdotnet.com/ team/ xml_wsspecs/ xlang-c/.

[29] A. Willig, M. Kubisch, C. Hoene, and A. Wolisz. Measurement of a wireless link in an industrial environment. *IEEE Transactions on Industrial Electronics*, 43(6):1265–1282, 2002.

[30] D. Wischik. *Large Deviations and Internet Congestion*. PhD thesis, University of Cambridge, 1999.

[31] L. Wischik. Pi implementation (1): Old names for nu, 2003. Submitted for publication.

[32] P. Wojciechowski and P. Sewell. Nomadic pict: Language and infrastructure design for mobile agents. *IEEE Concurrency*, 8(2):42–52, 2000.