

# Notes for the Verification of the Session Management Protocol

Karl Palmskog  
(palmskog@kth.se)

## Abstract

Notes about, and references to, relevant literature for a Master's Thesis  
Project at Ericsson AB.

## Contents

<b>1</b>	<b>Process algebra</b>	<b>2</b>
1.1	Overview . . . . .	2
1.2	$\pi$ -calculus . . . . .	2
<b>2</b>	<b>Programming language semantics</b>	<b>2</b>
<b>3</b>	<b>Modal and temporal logics</b>	<b>3</b>
<b>4</b>	<b>Model checking theory for processes</b>	<b>3</b>
<b>5</b>	<b>Protocol verification and analysis</b>	<b>3</b>
<b>6</b>	<b>Tools for process verification and analysis</b>	<b>4</b>
<b>7</b>	<b>Feature interaction</b>	<b>4</b>
<b>8</b>	<b>Checkpoint-based Data Consistency Protocol</b>	<b>5</b>
8.1	Service provisions . . . . .	5
8.2	Assumptions about the environment . . . . .	5
8.3	Vocabulary of protocol messages . . . . .	5
8.4	Message format . . . . .	5
8.5	Procedure rules . . . . .	6
<b>9</b>	<b>Websites</b>	<b>6</b>
	<b>References</b>	<b>7</b>

# 1 Process algebra

**Motivation:** provides methods for formally modeling concurrent systems.

## 1.1 Overview

On process theory, van Glabbeek [CITE] writes:

A *process* is the behaviour of a system. [...] Process theory is the study of processes.

On process algebra from [wikipedia.org](http://wikipedia.org):

In computer science, [process algebras] are a diverse family of related approaches to formally modelling concurrent systems.

Started with CCS/CSP — notion formalized by Bergstra and Klop et al [1]. States are usually pieces of syntax.

The reference work on CCS is [12]. Milner explains his approach to process algebra and communicating systems in [13].

The original paper on CSP is [8].

## 1.2 $\pi$ -calculus

The first two papers on  $\pi$ -calculus are [15], written in 1989. Now of mainly historical interest.

Milner wrote the first textbook on  $\pi$ -calculus [14] — it is a good introductory text, which requires considerable reader effort to understand fully. It has advanced and enlightening examples of the uses of  $\pi$ , e.g. expressing object-oriented programs. Another introductory text is [19].

An interesting paper which explores translation of  $\pi$ -calculus processes into PROMELA for SPIN verification is [23].

Sangiorgi's PhD thesis [21] shows how the  $\pi$ -calculus processes can transmit not only names, but also other processes —  $\pi$  is not restricted to “link mobility”.

The most comprehensive book on  $\pi$  to date is [22].

# 2 Programming language semantics

Three approaches:

- Operational semantics — executions on abstract machines; used for CCS/ $\pi$ .
- Denotational — maps programs to mathematical objects (functions for sequential programs); one attempt at denotational semantics for concurrent programs is Plotkin's powerdomain construction [CITE].
- Axiomatic — embed correctness assertions in the program, prove them using axioms; use e.g. Owicki-Gries for parallelism.

### 3 Modal and temporal logics

**Motivation:** used for formal specification of properties (such as liveness, termination) for processes, or any other transition systems — properties can then be checked algorithmically.

Modal logics and  $\mu$ -calculi are used for specifying properties of labeled transition systems, which includes ordinary sequential programs as well as communicating mobile systems. There are  $\mu$ -calculi variations for both  $\pi$  and CCS. Temporal logics, which apply to unlabeled transitions systems, are used for CSP (e.g. for PROMELA processes in the SPIN tool).

In understanding modal logics and  $\mu$ -calculi, knowledge of first-order logic, classical modal logic and automata theory is helpful [20, p. 11].  $\mu$ -calculus and its variation makes use of fixpoints to allow expression of safety and liveness properties in labeled transition systems. To work unhindered with fixpoints, a grasp of denotational semantics of programming languages and domain theory is useful. Chapters 5, 8 and 14 in [29] describe the underlying theory. [29, pp. 327ff] shows the  $\mu$ -calculus proof system for CCS by Stirling and Walker [25], in a form that is suitable for manual use.

A comprehensive, but advanced and demanding, introductory text on  $\mu$ -calculus is [4]. It describes the history, motivation and theory in a structured way, and explores the connections to other fields in computer science and mathematics such as automata theory and game theory.

$\mu$ -calculus in its present form was first described by Kozen in 1983 [11].

There is a connection between checking modal logic properties of processes and checking bisimulation between processes. The connection is explored in [16].

### 4 Model checking theory for processes

**Motivation:** used for processing formal specification and deciding whether properties hold or not

Many proof systems for automatic checking of modal properties of processes have been developed and included in tools. For example, the model checking system for  $\pi$ -calculus processes described in [6] was integrated into the Mobility Workbench (MWB) [26]. See also [2] for a sequent-calculus based approach to model checking in MWB.

Other papers on  $\pi$ -calculus model checking include [7] and [24].

### 5 Protocol verification and analysis

**Motivation:** heuristics and examples of approaches to verification of protocols and other concurrent systems.

CCS, the  $\pi$ -calculus precursor, has been successfully used for protocol verification. One example is [18]. [27] describes the use of CCS for the analysis of mutual exclusion algorithms.

Gerard Holzmann, the author of the SPIN verification tool (which has process syntax/semantics loosely based on CSP) has written two books on protocol verification: “Design and Validation of Computer Protocols” [9] and “The SPIN Model Checker — Primer and Reference Manual” [10].

Mitchell et al. [17] provides a general methodology for formal analysis of cryptographic protocols, with examples of its application.

Sewell et al [CITE] have produced formal descriptions of TCP from the informal specification. It is provided in the language of the theorem prover HOL [CITE].

## 6 Tools for process verification and analysis

- The Edinburgh Concurrency Workbench (modelling of CCS processes and model checking with the  $\mu$ -calculus) [5]  
<http://homepages.inf.ed.ac.uk/perdita/cwb/>
- The Mobility Workbench (modelling of  $\pi$ -calculus processes and model checking with the  $\pi$ - $\mu$ -calculus) [26]  
<https://www.it.uu.se/research/group/mobility/mwb>
- ProVerif (a cryptography protocol verifier which accepts input in the syntax of the Applied  $\pi$ -calculus) [3]  
<http://www.di.ens.fr/~blanchet/crypto-eng.html>
- SPIN — a widely deployed model checker  
<http://www.spinroot.com>

## 7 Feature interaction

A definition by Zave [CITE, p. 3]:

A feature interaction is a case in which system behavior as a whole does not satisfy the separate specifications of all its features. A feature interaction arises when the feature-by-feature specification of a systems is incomplete, ambiguous, or incorrect.

Feature interaction can be handled both formally and pragmatically. One formal approach concerns service specification in temporal logic — see Jonsson et al [CITE].

## 8 Checkpoint-based Data Consistency Protocol

A protocol which enables two hosts to agree on a checkpoint in their respective send and receive streams, from which it is possible to re-establish a session after a connection is lost [28]. We follow the protocol specification structure from Holzmann [9, pp. 21ff].

### 8.1 Service provisions

The purpose of the protocol is to let two hosts  $A$  and  $B$  continually agree on a checkpoint  $(s_A, s_B)$ , such that  $s_A$  ( $s_B$ , respectively) is the position in  $A$ 's ( $B$ 's) send stream up to which  $B$  ( $A$ ) is guaranteed to have received all data.

### 8.2 Assumptions about the environment

The protocol messages are guaranteed, through the use of a transport layer control protocol such as TCP, to arrive in the order they are sent and without errors. A message may be lost, but after the loss, no further messages will be delivered. The protocol executes in the context of an established transport layer connection.

### 8.3 Vocabulary of protocol messages

There are two types of messages: *cp* checkpoint messages which contain information on received and sent data from the source host, and *data* messages which contain application-related payload. The vocabulary may be expressed as a set:

$$V = \{cp, data\}.$$

### 8.4 Message format

A protocol message has four fields:

- Message ID — the message identification field, which indicates the message type.
- Checkpoint ID — the checkpoint identification field, which indicates which checkpoint the given information pertains to.
- Payload size — indicates the size of the payload, in bytes.
- Payload — the payload, which is divided into two subfields as follows if it is a checkpoint message:
  - Recieve stream position.
  - Send stream position.

## 8.5 Procedure rules

Both hosts maintain pointers to an acknowledged checkpoint and a pending checkpoint. They also keep track of the number of bytes they have sent and received.

When a receive buffer is full to a certain degree, a host creates a new checkpoint which it sends to its peer as a checkpoint message. If it receives a reply, it sets the pending checkpoint as the acknowledged checkpoint, and the new checkpoint as the pending checkpoint.

If a host receives an unsolicited checkpoint message, it first sets the pending checkpoint as the acknowledged checkpoint, and the new checkpoint as the pending checkpoint. It then replies with the new checkpoint.

A checkpoint message reflects what the sender currently knows about the receiver. Hence, the receiver will update a checkpoint with its latest send/receive values when it replies.

The positions in the streams are continually updated as data arrives. When data has been acknowledged, the acknowledged amount is subtracted from the counters.

Checkpoint messages are sent both as requests for checkpoint establishment and as response to requests.

## 9 Websites

- <http://move.to/mobility/> — calculi for mobile processes resource page
- <http://lampwww.epfl.ch/mobility/bib/honda.html> — selected Bibliography on Mobile Processes
- <http://www.afm.sbu.ac.uk/> – general information on formal methods

## References

- [1] J. C. M. Baeten and W. P. Weijland. *Process algebra*. Cambridge University Press, New York, NY, USA, 1990.
- [2] F. B. Beste. The model prover — a sequent-calculus based modal  $\mu$ -calculus model checker tool for finite control  $\pi$ -calculus agents, 1998.  
<http://www.it.uu.se/profundis/mwb-dist/x4.ps.gz>.
- [3] B. Blanchet. Automatic Verification of Cryptographic Protocols: A Logic Programming Approach (invited talk). In *5th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'03)*, pages 1–3, Uppsala, Sweden, Aug. 2003. ACM.  
<http://www.di.ens.fr/~blanchet/publications/BlanchetPPDP03.pdf>.
- [4] J. Bradfield and C. Stirling. *Handbook of Process Algebra*, chapter Modal logics and mu-calculi: an introduction. Elsevier, 2001.  
<http://www.dcs.ed.ac.uk/home/jcb/Research/bradfield-stirling-HPA-mu-intro.ps.gz>.
- [5] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: a semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15(1):36–72, 1993.
- [6] M. Dam. Model checking mobile processes. *Journal of Information and Computation*, 129(1):35–51, 1996. Also available as Research Report R94:01, SICS, Sweden. A summary appeared in *Proceedings of CONCUR '93*, LNCS 715. <http://web.it.kth.se/~mfd/Papers/mcmpiac.pdf>.
- [7] M. Dam. Proof systems for pi-calculus logics. In R. de Queiroz, editor, *Logic for Concurrency and Synchronisation*, Studies in Logic and Computation. Oxford University Press, 2001.  
<http://www.imit.kth.se/~mfd/Papers/pspcl.pdf>.
- [8] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 26(1):100–106, 1983.
- [9] G. J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Upper Saddle River, NJ, USA, 1991.  
<http://spinroot.com/spin/Doc/Book91.html>.
- [10] G. J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2003.  
[http://spinroot.com/spin/Doc/Book\\_extras/index.html](http://spinroot.com/spin/Doc/Book_extras/index.html).
- [11] D. Kozen. Results on the propositional  $\mu$ -calculus. *Theoret. Comput. Sci.*, 1(27):333–354, 1983.
- [12] R. Milner. *Communication and concurrency*. Prentice Hall, Hertfordshire, UK, 1989.
- [13] R. Milner. Elements of interaction: Turing award lecture. *Commun. ACM*, 36(1):78–89, 1993.
- [14] R. Milner. *Communicating and Mobile Systems: the  $\pi$ -Calculus*. Cambridge University Press, May 1999.

- [15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, Sept. 1992. <http://www.lfcs.informatics.ed.ac.uk/reports/89/ECS-LFCS-89-85/>.
- [16] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Comput. Sci.*, 114:149–171, 1993. <http://www.it.kth.se/~joachim/modmob.ps.gz>.
- [17] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi, 1997. <http://sprout.stanford.edu/dill/PAPERS/verification/MMS97.ps>.
- [18] J. Parrow. Verifying a CSMA/CD-protocol with CCS. In *Proceedings of the IFIP WG6.1 Eighth International Symposium on Protocol Specification, Testing, and Verification*, pages 373–384, June 1988. <http://www.imit.kth.se/courses/2G1516/Docs05/CCS/CSMA-CD-Parrow.pdf>.
- [19] J. Parrow. *Handbook of Process Algebra*, chapter An introduction to the pi-Calculus, pages 479–543. Elsevier, 2001. <http://user.it.uu.se/~joachim/intro.ps>.
- [20] D. A. Peled. *Software reliability methods*. Springer-Verlag New York, Secaucus, NJ, USA, 2001.
- [21] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1993. CST-99-93 (also published as ECS-LFCS-93-266).
- [22] D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
- [23] H. Song and K. J. Compton. Verifying  $\pi$ -calculus processes by Promela translation. *Technical Report 472, University of Michigan*, 2003. <http://www.eecs.umich.edu/techreports/cse/03/CSE-TR-472-03.pdf>.
- [24] C. Sprenger and M. Dam. A note on global induction mechanisms in  $\mu$ -calculus with explicit approximations. *Theoretical Informatics and Applications*, 1(37):365–399, 2003. <http://www.imit.kth.se/~mfd/Papers/ITA-final.pdf>.
- [25] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *TAPSOFT '89: 2nd international joint conference on Theory and practice of software development*, pages 161–177, Amsterdam, The Netherlands, 1991. Elsevier Science Publishers B. V.
- [26] B. Victor. *A Verification Tool for the Polyadic  $\pi$ -Calculus*. Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50. <http://www.docs.uu.se/~victor/tr/docs-tr-94-50.html>.
- [27] D. Walker. Analysing mutex exclusion algorithms using ccs. *Formal Aspects of Computing*, 1:273–292, 1989.

- [28] M. Widell and P. Arvidsson. Design of the session layer supporting mobile, delay and disconnection tolerant communication. Master's thesis, KTH, 2006. To appear.
- [29] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.