# The Mobility Workbench*
# — A Tool for the $\pi$-Calculus —

Björn Victor† Faron Moller‡

**Abstract**

In this paper we describe the first prototype version of the Mobility Workbench (MWB), an automated tool for manipulating and analyzing mobile concurrent systems (those with evolving connectivity structures) described in the $\pi$-calculus. The main feature of this version of the MWB is checking open bisimulation equivalences. We illustrate the MWB with an example automated analysis of a handover protocol for a mobile telephone system.

*Dedicated to Ellen on the occasion of her birth.*

## 1 Introduction

*Process algebra* is the general study of distributed concurrent systems in an algebraic framework. There have been many successful models formulated within this framework, one representative example being Milner's CCS [10]. Each approach has added to more than a decade of fruitful discoveries on the mathematical foundations of concurrent processes, so that now it is the case that these theories can be applied in practice, perhaps using automated tools of which there are many; for a useful survey see [9]. Certainly, as systems become more complex, it becomes necessary to invoke the use of automated tools to aid in their analyses. These tools however exploit the fact that properties of *finite-state* systems are decidable, and hence they cannot be used except for this simple class of systems.

A shortcoming of these process algebras, which was perhaps necessary for the development of such a complex field, is that they enforce restrictions on the nature of the systems which they attempt to model. One such restriction is the inability to model evolving communication structures. However this particular shortcoming is now being tackled within the CCS framework with the $\pi$-calculus [13], an extension of CCS which allows for the modelling of *mobility* within systems, the ability for systems to dynamically alter their communication structures. The foremost problem with such an extension is the greatly increased complexity of the analysis of systems. One may say that our understanding of finite-state systems is rather complete now; however, with extra constructs within the algebra it becomes nontrivial to even define, let alone to then decide, semantic equivalences between systems.

---

*Research supported by ESPRIT BRA Grants 6454: CONFER and 7166: CONCUR2.

†Dept of Computer Systems, Uppsala University, Box 325, S-751 05 Uppsala, Sweden, and Swedish Institute for Computer Science.

‡Dept of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh.

In this paper we describe the MWB (Mobility Workbench), a tool for manipulating and analyzing mobile concurrent systems described in the $\pi$-calculus. In the current version, the basic functionality is to decide the *open bisimulation equivalences* of Sangiorgi [16], for agents in the monadic $\pi$-calculus with the original positive match operator. This is decidable for $\pi$-calculus agents with *finite control* — analogous to CCS finite-state agents — which do not admit parallel composition within recursively defined agents. There are various other analysis routines implemented, including commands for finding deadlocks and for interactively simulating agents.

The outline of the paper is as follows. We start in Section 2 with a brief presentation of the $\pi$-calculus, its syntax and semantic definition, as well as the definition of equality of agents. We also briefly describe aspects of the implementation of equality checking. We then present the MWB in Section 3, demonstrating its main utilities on simple examples. In Section 4 we describe an extended realistic case study, the automated verification of a part of a mobile telephone protocol. Finally in Section 5 we describe future development plans for the MWB.

## 2 Mobile Processes: The $\pi$-calculus

In this section we give a brief presentation of the syntax and semantics of the $\pi$-calculus, as well as a description of the open bisimulation equivalences and efficient characterisations for these which will be used in the implementation. For fuller treatments of these topics we refer to [13, 16].

There are two entities in the $\pi$-calculus: *names* (ranged over by $x$,$y$,$z$,$w$,$v$,$u$), and *processes* (ranged over by $P$,$Q$,$R$). The syntax of the $\pi$-calculus is given by the following BNF equation

$$P \ ::= \ \mathbf{0} \ \big| \ A(x_1,\ldots,x_k) \ \big| \ \alpha.P \ \big| \ [x=y]P \ \big| \ P_1+P_2 \ \big| \ P_1|P_2 \ \big| \ (\boldsymbol{\nu}x)P$$

where $A$ ranges over some set of variables with associated nonnegative arities $k$, and $\alpha$ represents an input $x(y)$, a free output $\overline{x}y$, a bound output $\overline{x}(y)$, or a silent event $\tau$. Briefly, $\mathbf{0}$ represents an inactive process; each process variable $A(x_1,\ldots,x_k)$ has a corresponding definitional body $P$; matching $[x=y]P$ is read as *"if $x=y$ then $P$"*; sum $P_1+P_2$ offers the choice of $P_1$ or $P_2$; composition $P_1|P_2$ places the two processes $P_1$ and $P_2$ side-by-side in parallel execution; restriction $(\boldsymbol{\nu}x)P$ hides the name $x$ from the environment of $P$; and action prefixing $\alpha.P$ performs the relevant input, output or silent transition, thus evolving into $P$. (Bound output $\overline{x}(y).P$ is in fact simply shorthand for the expression $(\boldsymbol{\nu}y)\overline{x}y.P$.)

The definitions of *free* and *bound* names are standard ($x(y).P$ and $(\boldsymbol{\nu}y)P$ bind $y$), and we shall write $fn(P)$ and $fn(\alpha)$ for the free names of $P$ and $\alpha$; $bn(P)$ and $bn(\alpha)$ for the bound names of $P$ and $\alpha$; and $n(P)$ and $n(\alpha)$ for the names of $P$ and $\alpha$. The definitions for substitution and alpha conversion are equally standard,

$$\mathsf{pre}: \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \mathsf{match}: \frac{P \xrightarrow{\alpha} P'}{[x=x]P \xrightarrow{\alpha} P'}$$

$$\mathsf{Ide}: \frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'} \left(A(\tilde{x}) \stackrel{\mathrm{def}}{=} P\right) \qquad \mathsf{sum}: \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$

$$\mathsf{par}: \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \left(bn(\alpha) \cap fn(Q) = \emptyset\right) \qquad \mathsf{com}: \frac{P \xrightarrow{\overline{x}y} P', \ \ Q \xrightarrow{x(z)} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{y/z\}}$$

$$\mathsf{open}: \frac{P \xrightarrow{\overline{x}y} P'}{(\boldsymbol{\nu} y)P \xrightarrow{\overline{x}(y)} P'} \left(x \neq y\right) \qquad \mathsf{close}: \frac{P \xrightarrow{\overline{x}(y)} P', \ \ Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\tau} (\boldsymbol{\nu} y)\left(P'|Q'\right)}$$

$$\mathsf{res}: \frac{P \xrightarrow{\alpha} P'}{(\boldsymbol{\nu} x)P \xrightarrow{\alpha} (\boldsymbol{\nu} x)P'} \left(x \notin n(\alpha)\right)$$

Figure 1: Semantic derivation rules.

with renaming possible to avoid name capture. We shall always identify processes or transitions which differ only on bound names.

In Figure 1 we present the operational rules for the $\pi$-calculus. We have omitted symmetric rules for sum, par, com and close. From this definition of the single-step transition system we can define the weak transition system which abstracts from silent transitions in the usual fashion: $\Longrightarrow$ represents $(\xrightarrow{\tau})^*$ and $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ represents $\Longrightarrow$ when $\alpha = \tau$ and $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$ when $\alpha \neq \tau$.

## 2.1 Open Bisimulation

There are several ways in which one can define bisimilarity of $\pi$-calculus terms, varying on the attitude taken towards name instantiation. Notable among these are the *early* and *late* bisimulations of [13]. We choose to concentrate on the elegant notion of *open bisimilarity* of [16] — which is finer than the previous equivalences — for several reasons. Firstly, the strong version is a congruence. In particular, it is preserved by input prefix, unlike either of early or late equivalence. Its naturality is attested to by a simple axiomatisation. Most importantly, it has an efficient characterisation (described below) which we exploit in the implementation.

**Definition 2.1** *A binary relation $\mathcal{R}$ between process terms is a* strong open bisimulation *if whenever $(P, Q) \in \mathcal{R}$ then for each substitution $\sigma$ from names to names,*

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

$P$ *and* $Q$ *are* strongly open bisimilar*, written* $P \sim Q$*, if* $(P, Q) \in \mathcal{R}$ *for some strong open bisimulation* $\mathcal{R}$*.*

*If we replace* $\xrightarrow{\alpha}$ *by* $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ *in the consequents of the two clauses in the above definition, then* $P$ *and* $Q$ *are* (weak) open bisimilar*, written* $P \approx Q$*.*

This definition is actually only adequate for the calculus without restriction. The inclusion of restriction requires a treatment of *distinctions* [13], symmetric and irreflexive binary relations over names stipulating when names are not allowed to be equated by instantiation. We can however define open bisimilarity with respect to distinctions; these will be referred to by $\sim_D$ and $\approx_D$ for the strong and weak relations, respectively. For example, $[x = y]z.\mathbf{0} \sim_{\{(x,y)\}} \mathbf{0}$ though these two terms are not equal by the original definition of $\sim$. We do not go into these definitions in detail here, leaving the reader instead to consult [13, 16], but we note that the open bisimilarity relations correspond to the cases when the distinctions $D$ are empty.

## 2.2 An Efficient Characterisation of Open Bisimulation

The definition of open bisimilarity (as with those for early and late equivalence) involves universal quantifications over substitutions, which make a direct implementation infeasible. However, there is an alternate characterisation for open bisimilarity based on the transition system presented in Figure 2 which is similar to the approach of [8]. The transitions here are of the form $P \stackrel{M,\alpha}{\rightsquigarrow} P'$, where $M$ intuitively represents the least condition (set of name identities) under which action $\alpha$ can occur. Thus for example we have $[x = y]\alpha.P \stackrel{x=y,\alpha}{\rightsquigarrow} P$. These composite transitions $(M, \alpha)$ are ranged over by $\mu$, and we write $n(M, \alpha)$ for $n(M) \cup n(\alpha)$ and similarly for $bn(M, \alpha)$. We shall always assume that the condition $x = x$ is ignored in the rules match, com and close, so that for example $[x = x]P \stackrel{\mu}{\rightsquigarrow} P'$ whenever $P \stackrel{\mu}{\rightsquigarrow} P'$.

We also define a weak transition system: $\stackrel{M,\alpha}{\approx\!\!\!>}$ represents $(\stackrel{N_1,\tau}{\rightsquigarrow} \cdots \stackrel{N_n,\tau}{\rightsquigarrow}) \stackrel{L,\alpha}{\rightsquigarrow} (\stackrel{K_1,\tau}{\rightsquigarrow} \cdots \stackrel{K_m,\tau}{\rightsquigarrow})$ for $n, m \geq 0$, where $M = L \wedge \bigwedge_i N_i \wedge \bigwedge_i K_i$ if $\alpha \neq \tau$, and $(\stackrel{N_1,\tau}{\rightsquigarrow} \cdots \stackrel{N_n,\tau}{\rightsquigarrow})$ for $n \geq 0$, where $M = \bigwedge_i N_i$ if $\alpha = \tau$, where in either case no name bound in $\alpha$ occurs in the accumulated condition $M$.

In the following definition, we denote by $\sigma_M$ the substitution on names induced by the equivalence classes associated with the equivalence relation corresponding to $M$; we select one representative of each class and map the other members of the class to it. We also use $\Rightarrow$ to denote logical implication, and $\equiv$ to denote equality modulo alpha conversion.

$$\mathsf{pre}: \frac{}{\alpha.P \stackrel{\mathsf{true},\alpha}{\leadsto} P} \qquad \mathsf{match}: \frac{P \stackrel{M,\alpha}{\leadsto} P'}{[x=y]P \stackrel{M \wedge x=y,\alpha}{\leadsto} P'} \Big(x, y \notin bn(\alpha)\Big)$$

$$\mathsf{Ide}: \frac{P\{\tilde{y}/\tilde{x}\} \stackrel{\mu}{\leadsto} P'}{A(\tilde{y}) \stackrel{\mu}{\leadsto} P'} \Big(A(\tilde{x}) \stackrel{\mathrm{def}}{=} P\Big) \qquad \mathsf{sum}: \frac{P \stackrel{\mu}{\leadsto} P'}{P + Q \stackrel{\mu}{\leadsto} P'}$$

$$\mathsf{par}: \frac{P \stackrel{\mu}{\leadsto} P'}{P|Q \stackrel{\mu}{\leadsto} P'|Q} \Big(bn(\mu) \cap fn(Q) = \emptyset\Big) \qquad \mathsf{com}: \frac{P \stackrel{M,\overline{x}y}{\leadsto} P', \ Q \stackrel{N,w(z)}{\leadsto} Q'}{P|Q \stackrel{M \wedge N \wedge x=w,\tau}{\leadsto} P'|Q'\{y/z\}}$$

$$\mathsf{open}: \frac{P \stackrel{M,\overline{x}y}{\leadsto} P'}{(\boldsymbol{\nu} y)P \stackrel{M,\overline{x}(y)}{\leadsto} P'} \Big(y \notin n(M) \cup \{x\}\Big) \qquad \mathsf{close}: \frac{P \stackrel{M,\overline{x}(y)}{\leadsto} P', \ Q \stackrel{N,w(y)}{\leadsto} Q'}{P|Q \stackrel{M \wedge N \wedge x=w,\tau}{\leadsto} (\boldsymbol{\nu} y)\Big(P'|Q'\Big)}$$

$$\mathsf{res}: \frac{P \stackrel{\mu}{\leadsto} P'}{(\boldsymbol{\nu} x)P \stackrel{\mu}{\leadsto} (\boldsymbol{\nu} x)P'} \Big(x \notin n(\mu)\Big)$$

Figure 2: Alternate transition system.

**Definition 2.2** *A binary relation $\mathcal{R}$ between process terms is a* strong $\asymp$-bisimulation *if whenever $(P, Q) \in \mathcal{R}$ then*

- *if $P \stackrel{M,\alpha}{\leadsto} P'$ then $Q \stackrel{N,\beta}{\leadsto} Q'$ for some $N$, $\beta$ and $Q'$ with $M \Rightarrow N$, $\alpha\sigma_M \equiv \beta\sigma_M$ and $(P'\sigma_M, Q'\sigma_M) \in \mathcal{R}$;*
- *if $Q \stackrel{M,\alpha}{\leadsto} Q'$ then $P \stackrel{N,\beta}{\leadsto} P'$ for some $N$, $\beta$ and $P'$ with $M \Rightarrow N$, $\alpha\sigma_M \equiv \beta\sigma_M$ and $(P'\sigma_M, Q'\sigma_M) \in \mathcal{R}$.*

*$P$ and $Q$ are* strongly $\asymp$-bisimilar*, written $P \asymp Q$, if $(P, Q) \in \mathcal{R}$ for some strong $\asymp$-bisimulation $\mathcal{R}$.*

*If we replace $\stackrel{N,\beta}{\leadsto}$ by $\stackrel{N,\beta}{\approx\!\!>}$ in the above definition, then $P$ and $Q$ are* (weak) $\asymp$-bisimilar*, written $P \mathrel{\overset{\frown}{\asymp}} Q$.*

Again this definition is valid only for the subcalculus without restriction, but again we can define $\asymp$-bisimilarity with respect to distinctions, and again we leave the details to [16].

The following theorems from [16] and [18] respectively, are what we are particularly interested in.

**Theorem 2.3 (Sangiorgi)** *$\sim_D$ coincides with $\asymp_D$*

**Theorem 2.4 (Victor)** *$\approx_D$ coincides with $\mathrel{\overset{\frown}{\asymp}}_D$.*

### 2.3 Algorithmic Aspects

The usual technique for deciding bisimilarity of (finite-state) systems is to construct the state space of the two systems in question and then perform a partition refinement algorithm to try to distinguish the two start states. However, this technique is inapplicable in the case of the $\pi$-calculus due to the problems of name instantiation.

For example, the two terms

$$P(x) \stackrel{\text{def}}{=} x(y).\overline{y}y.P(y) \qquad\qquad Q(x) \stackrel{\text{def}}{=} x(y).\Big(\overline{y}y.Q(y) \;+\; [x=y]\overline{y}y.Q(y)\Big)$$

are clearly open bisimilar. In contrast with the late and early equivalences, with $\asymp$-equivalence it is enough to instantiate the bound name of an input with a single fresh name (a name $x$ is *fresh* with respect to a transition $P \stackrel{\mu}{\rightsquigarrow} Q$ if $x \notin fn(Q) - bn(\mu)$). But then $P(x)$ has a minimal state space consisting of only two states, namely itself and the intermediate state $\overline{x}x.P(x)$ attained by instantiating the $y$ by $x$, thus performing the input action $x(x)$. Regardless of how the state space of $Q(x)$ is generated, it cannot be equated to the above state space of $P(x)$, as its first transition cannot instantiate $y$ to $x$, due to the appearance of $x$ in the ensuing process. To match the two, the state space of $P(x)$ must be extended to match that of $Q(x)$.

Thus the implementation of the bisimulation algorithm is by necessity an *"on-the-fly"* algorithm [4]: the state spaces of the two systems in question are generated together during the construction of the candidate bisimulation relation which equates them.

Beyond this, the algorithm implemented in the MWB follows Definition 2.2 (both its strong and weak versions) closely: given two agents $P$ and $Q$ and a relation $\mathcal{R}$, check if $(P,Q)$ is already in the relation. If so, return the relation unchanged. Otherwise, for each transition of $P$, find a (strong or weak) transition of $Q$ such that the conditions match appropriately and the actions are equivalent under the substitution $\sigma$ induced by the first, larger, condition. Make the transitions instantiate the same bound name (alpha-converting the derivatives), and assuming that $P$ and $Q$ are equivalent (by adding $(P,Q)$ to the relation), apply the substitution $\sigma$ to the derivatives and recurse over them using the extended relation. This either fails, which causes the current recursion to try the next transition of $Q$ (or to fail if no such transition exists), or returns a relation relating the two derivatives, which is used in subsequent recursive calls. Finally, when all transitions of $P$ have been matched by $Q$, match each transition of $Q$ with a (strong or weak) transition of $P$ in the same way, returning the resulting relation. (This is the approach in the absence of distinctions; in the more general case, distinctions are handled in a suitable fashion.)

## 3 The Mobility Workbench

The basic functionality of the MWB is to decide (strong and weak) open bisimilarity. Amongst other things, it can also be used to find deadlocks and for interactively simulating agents.

In Figure 3 we have a sample session which demonstrates some simple usage. Note that we render $\boldsymbol{\nu}$ as ^ and $\overline{x}y$ as 'x<y> when typing in ASCII format. First we define an agent Buf1 implementing a one-place buffer, then another, Buf2, implementing a two-place buffer by composing two instances of Buf1, and finally three agents, Buf20, Buf21 and Buf22, together implementing a two-place buffer without parallel composition.

We proceed with this example by comparing the two implementations for weak equality. The MWB responds by saying that they are equivalent and that it found a bisimulation relation with 18 tuples, and asks us if we want to inspect it. We respond positively and the MWB prints out the relation as a list of pairs of agents with associated distinction sets.

We then simulate the behaviour of the agent Buf2(i,o). The MWB presents the possible transitions, along with their least necessary conditions (if not trivial), and prompts the user to select one of them. After having a single choice on the first two steps, we then get a choice of three transitions; the first which is possible only if the names i and o are the same; the second which uses a new name ~v0 since all other known names are free and thus can't be reused; and the third, which simply outputs the value we read.

Next, we change the definition of Buf22 to introduce a possible deadlock and again check for weak equivalence between Buf2(i,o) and Buf20(i,o). This time we find that they are not equivalent, and proceed by looking for deadlocks in Buf20(i,o); as the MWB finds deadlocked agents, it tells us the agent and the transition trace that leads to the deadlocked agent.

Finally we try equating Buf2(i,o) and Buf20(i,o) under the proviso that i is different from all other free names of the two agents (namely o). Under this distinction, the deadlocks don't appear, and the MWB reports that they are again equivalent.

## 4 An Extended Example: Mobile Telephones

As a case study, we have specified and verified the core of the handover protocol intended to be used in the GSM Public Land Mobile Network (PLMN) proposed by the European Telecommunication Standards Institute (ETSI). The formal specification of the protocol, and its service specification, are due to Orava and Parrow [14], who also verified the protocol algebraically. Fredlund and Orava [5] later verified the protocol automatically by specifying the protocol in LOTOS [17], which was translated to labelled transition systems using the Cæsar tool [6], which were in turn minimized using the Aldébaran tool [3], and finally compared

```
The Mobility Workbench
(Preliminary version 0.86, built Tue Oct 12 15:27:55 MET 1993)


MWB> agent Buf1(i,o) = i(x).'o<x>.Buf1(i,o)
MWB> agent Buf2(i,o) = (^m)(Buf1(i,m) | Buf1(m,o))
MWB> agent Buf20(i,o) = i(x).Buf21(i,o,x)
MWB> agent Buf21(i,o,x) = i(y).Buf22(i,o,x,y) + 'o<x>.Buf20(i,o)
MWB> agent Buf22(i,o,x,y) = 'o<x>.Buf21(i,o,y)

MWB> weq Buf2(i,o) Buf20(i,o)
The two agents are related.
Relation size = 18.  Do you want to see it?  (y or n) y
R = < Buf2(i,o),  Buf20(i,o) >  {}
    < (^y)(Buf1(i,y) | Buf1(y,o)),  Buf20(i,o) >  {}
    < (^y)('y<m>.Buf1(i,y) | 'o<x>.Buf1(y,o)),  Buf22(i,o,x,m) >  {}
      . . .

MWB> step Buf2(i,o)
   0:  -- i(x) --> (^m)('m<x>.Buf1(i,m) | Buf1(m,o))
Step> 0
   0:  -- t --> (^m)(Buf1(i,m) | 'o<x>.Buf1(m,o))
Step> 0
   0:  -- [i=o],t --> (^m)('m<x>.Buf1(i,m) | Buf1(m,o))
   1:  -- i(~v0) --> (^m)('m<~v0>.Buf1(i,m) | 'o<x>.Buf1(m,o))
   2:  -- 'o<x> --> (^m)(Buf1(i,m) | Buf1(m,o))
Step> 1
   0:  -- 'o<x> --> (^m)('m<~v0>.Buf1(i,m) | Buf1(m,o))
Step> quit

MWB> agent Buf22(i,o,x,y) = 'o<x>.Buf21(i,o,y) + [i=o]t.0

MWB> weq Buf2(i,o) Buf20(i,o)
The two agents are NOT related.

MWB> deadlocks Buf20(i,o)
Deadlock found in 0, reachable by 3 transitions:
   -- i(x) -- i(y) -- [i=o],t -->
Deadlock found in 0, reachable by 5 transitions:
   -- i(x) -- i(y) -- 'o<x> -- i(~v0) -- [i=o],t -->
Deadlock found in 0, reachable by 7 transitions:
   -- i(x) -- i(y) -- 'o<x> -- i(~v0) -- 'o<y> -- i(y) -- [i=o],t -->

MWB> weqd (i) Buf2(i,o) Buf20(i,o)
The two agents are related.
Relation size = 8.  Do you want to see it?  (y or n) y
R = < Buf2(i,o),  Buf20(i,o) >  {i#o}
    < (^y)(Buf1(i,y) | Buf1(y,o)),  Buf20(i,o) >  {i#o}
    < (^y)('y<m>.Buf1(i,y) | 'o<x>.Buf1(y,o)),  Buf22(i,o,x,m) >  {i#o}
      . . .
```

Figure 3: A simple sample session with the MWB.

using Aldébaran. The following informal presentation of the protocol is based on the presentation in [5].

The PLMN is a cellular system which can be seen as consisting of Mobile Stations (MSs), Base Stations (BSs), and Mobile Switching Centres (MSCs). The MS, mounted in e.g. a car, provides service to an end user. The BS manages the interface between the MS and a stationary network, controlling all radio communication within a geographical area (a cell). All communication with the MS in a cell is routed through the BS responsible for the cell. The MSC manages a set of BSs, and communicates with them and with other MSCs using a stationary network.

When a MS moves across a cell boundary, the handover procedure changes the communication partner of the MS from the BS of the old cell to the BS of the new cell, ensuring that the MS is constantly in contact with the MSC. The MSC initiates the handover by transmitting a *handover command* message to the MS via the old BS. The handover command message contains parameters enabling the MS to locate the new BS. When transmitting this message the MSC suspends transmission of all messages except for messages related to the handover procedure. When the MS receives the handover command message, it disconnects the old radio links and initiates the new radio links. To establish these connections the MS sends *handover access* messages to the new BS, in order to synchronize with the new BS. When the connections are successfully established, the BS sends a *handover complete* message to the MSC via the new BS. When this message has been received, the network resumes normal operations and releases the old radio links, which are now free and can be allocated to another MS.

In Figure 4 we present a $\pi$-calculus specification of the protocol. This is drawn from [14], but in this presentation we omit the failure handling aspects of the protocol. In Figure 5 we present the MWB code for this specification, which differs from Figure 4 in that the argument lists of agent identifier definitions must contain all free names which appear in the agent.

Correctness of this specification would come from showing that it matched some (ideally simple) service specification which would clearly define the desired behaviour of the system. In Figures 6 and 7, we present the service specification of the handover protocol and its rendering into MWB code, respectively.

When checking the protocol specification (`System`) against the more abstract service specification (`Spec`), we must express the fact that the parameters `in`, `out`, `ho_acc`, `ho_com`, `data`, `ho_cmd` and `ch_rel` are constants, i.e. they are distinct from all other free names. This is done by using the `weqd` (weak open bisimulation with distinctions) command of the MWB, in the following way:

```
weqd (i,o,acc,com,data,cmd,rel)
     Spec(i,o)   System(i,o,acc,com,data,cmd,rel)
```

$$
\begin{array}{lcl}
CC(f_a, f_p, l) & \stackrel{\mathrm{def}}{=} & \;\;\mathtt{in}(v).\overline{f}_a\mathtt{data}.\overline{f}_a v.CC(f_a, f_p, l) \\
 & & +\, l(m_{new}).\overline{f}_a\mathtt{ho\_cmd}.\overline{f}_a m_{new}.f_p(c). \\
 & & \qquad [c = \mathtt{ho\_com}]\overline{f}_a\mathtt{ch\_rel}.f_a(m_{old}).\overline{l}m_{old}.CC(f_p, f_a, l) \\
HC(l, m) & \stackrel{\mathrm{def}}{=} & \overline{l}m.l(m).HC(l, m) \\
MSC(f_a, f_p, m) & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu} l)\Big(HC(l, m) \mid CC(f_a, f_p, l)\Big) \\
BS_a(f, m) & \stackrel{\mathrm{def}}{=} & f(c).\Big(\;[c = \mathtt{data}]f(v).\overline{m}\mathtt{data}.\overline{m}v.BS_a(f, m) \\
 & & \qquad +\, [c = \mathtt{ho\_cmd}]f(v).\overline{m}\mathtt{ho\_cmd}.\overline{m}v. \\
 & & \qquad\qquad\qquad f(c).[c = \mathtt{ch\_rel}]\overline{f}m.BS_p(f, m)\Big) \\
BS_p(f, m) & \stackrel{\mathrm{def}}{=} & m(c).[c = \mathtt{ho\_acc}]\overline{f}\mathtt{ho\_com}.BS_a(f, m) \\
MS(m) & \stackrel{\mathrm{def}}{=} & m(c).\Big(\;[c = \mathtt{data}]m(v).\overline{\mathtt{out}}v.MS(m) \\
 & & \qquad +\, [c = \mathtt{ho\_cmd}]m(m_{new}).\overline{m}_{new}\mathtt{ho\_acc}.MS(m_{new})\Big) \\
P(f_a, f_p) & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu} m)\Big(MSC(f_a, f_p, m) \mid BS_p(f_p, m)\Big) \\
Q(f_a) & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu} m)\Big(BS_a(f_a, m) \mid MS(m)\Big) \\
System & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu} f_a)(\boldsymbol{\nu} f_p)\Big(P(f_a, f_p) \mid Q(f_a)\Big)
\end{array}
$$

Figure 4: A formal specification of the handover procedure. After [14].

The bisimulation found by the MWB has 823 tuples. Running on a Sun SPARCstation 2 with 32Mb memory, it took approximately 61 CPU hours (user mode). The ML heap size was at most 17430 kb (well below the physical memory size of the machine). This appears to be rather extreme, and indeed the current prototype version of the MWB was not written with efficiency in focus. However, the new version of MWB being developed (see Section 5) finds a bisimulation with 249 tuples (leaving out alpha-equivalents) in just over 6 minutes CPU time on a SPARCstation 10, using 135 Mb heap space.

# 5 Future Development

On the theoretical side, the weak equivalence $\approx_D$ should be further investigated, e.g. regarding its axiomatization and its relationship to the late and early weak equivalences.

The prototype version of the MWB described here leaves room for many improvements. One deficiency is that it only handles the monadic $\pi$-calculus, where only one name can be sent or received atomically, while the polyadic $\pi$-calculus [11] generalises communication to allow zero or more names. The polyadic $\pi$-calculus

```
MWB> agent CC(fa,fp,l,in,data,ho_cmd,ho_com,ch_rel) =
       in(v).'fa<data>.'fa<v>.CC(fa,fp,l,in,data,ho_cmd,ho_com,ch_rel)
     + l(mnew).'fa<ho_cmd>.'fa<mnew>.fp(c).  [c=ho_com]'fa<ch_rel>.
                     fa(mold).'l<mold>.CC(fp,fa,l,in,data,ho_cmd,ho_com,ch_rel)

MWB> agent HC(l,m) = 'l<m>.l(m).HC(l,m)

MWB> agent MSC(fa,fp,m,in,data,ho_cmd,ho_com,ch_rel) =
       (^l)(HC(l,m) | CC(fa,fp,l,in,data,ho_cmd,ho_com,ch_rel))

MWB> agent BSa(f,m,ho_acc,ho_com,data,ho_cmd,ch_rel) =
       f(c).([c=data]f(v).'m<data>.'m<v>.BSa(f,m,ho_acc,ho_com,data,ho_cmd,ch_rel)
           + [c=ho_cmd]f(v).'m<ho_cmd>.'m<v>.f(c).
                     [c=ch_rel]'f<m>.BSp(f,m,ho_acc,ho_com,data,ho_cmd,ch_rel))

MWB> agent BSp(f,m,ho_acc,ho_com,data,ho_cmd,ch_rel) =
       m(c).[c=ho_acc]'f<ho_com>.BSa(f,m,ho_acc,ho_com,data,ho_cmd,ch_rel)

MWB> agent MS(m,data,ho_cmd,out,ho_acc) =
       m(c).( [c=data]m(v).'out<v>.MS(m,data,ho_cmd,out,ho_acc)
             +[c=ho_cmd]m(mnew).'mnew<ho_acc>.MS(mnew,data,ho_cmd,out,ho_acc))

MWB> agent P(fa,fp,in,ho_acc,ho_com,data,ho_cmd,ch_rel) =
       (^m)( MSC(fa,fp,m,in,data,ho_cmd,ho_com,ch_rel)
           | BSp(fp,m,ho_acc,ho_com,data,ho_cmd,ch_rel))

MWB> agent Q(fa,out,ho_acc,ho_com,data,ho_cmd,ch_rel) =
       (^m)( BSa(fa,m,ho_acc,ho_com,data,ho_cmd,ch_rel)
           | MS(m,data,ho_cmd,out,ho_acc))

MWB> agent System(in,out,ho_acc,ho_com,data,ho_cmd,ch_rel) =
       (^fa)(^fp)( P(fa,fp,in,ho_acc,ho_com,data,ho_cmd,ch_rel)
                 | Q(fa,out,ho_acc,ho_com,data,ho_cmd,ch_rel))
```

Figure 5: The MWB code for the protocol specification.

also allows a notion of sorts, roughly analogous to the notion of types in functional programming.

Another shortcoming of the current version of the MWB is of course its inefficiency. We are currently developing a new version of the MWB with efficiency as one of its goals; e.g. using de Bruijn indices [1] to represent names (making alpha conversion unnecessary), and using hash tables to record the possible transitions of an agent instead of computing them each time they are needed. This version will handle the polyadic $\pi$-calculus, and will include the sort inference algorithm developed by Gay [7]. The model checking algorithm due to Dam [2] is also being implemented. As hinted at in the previous section, this new version is providing great gains in efficiency. In particular, with the new version we are able to verify the full handover protocol as presented in [14] in less than 11 CPU minutes.

On the longer term, we expect the tool to evolve with the needs of users: adding more “utility” commands, e.g., for minimizing agents, finding distinguishing

$$
\begin{aligned}
Spec &\stackrel{\text{def}}{=} \mathtt{in}(v).S_1(v) \ + \ \tau.Spec \\
S_1(v) &\stackrel{\text{def}}{=} \mathtt{in}(v).S_2(v_1, v) \ + \ \overline{\mathtt{out}}v_1.Spec \ + \ \tau.\overline{\mathtt{out}}v_1.Spec \\
S_2(v_1, v_2) &\stackrel{\text{def}}{=} \mathtt{in}(v).S_3(v_1, v_2, v) \ + \ \overline{\mathtt{out}}v_1.S_1(v_2) \ + \ \tau.\overline{\mathtt{out}}v_1.\overline{\mathtt{out}}v_2.Spec \\
S_3(v_1, v_2, v_3) &\stackrel{\text{def}}{=} \overline{\mathtt{out}}v_1.S_2(v_2, v_3)
\end{aligned}
$$

Figure 6: A formal specification of the service specification.

```
MWB> agent Spec(in,out) = in(v).S1(v,in,out) + t.Spec(in,out)
MWB> agent S1(v1,in,out) =
         in(v).S2(v1,v,in,out) + 'out<v1>.Spec(in,out) + t.'out<v1>.Spec(in,out)
MWB> agent S2(v1,v2,in,out) =
         in(v).S3(v1,v2,v,in,out) + 'out<v1>.S1(v2,in,out) +
t.'out<v1>.'out<v2>.Spec(in,out)
MWB> agent S3(v1,v2,v3,in,out) = 'out<v1>.S2(v2,v3,in,out)

MWB> weqd (i,o,acc,com,data,cmd,rel) Spec(i,o) System(i,o,acc,com,data,cmd,rel)
The two agents are related.
Relation size = 823.  Do you want to see it?  (y or n) n
```

Figure 7: The MWB code for the service specification.

formulae, etc. We would also like to see graphical interfaces based on Sangiorgi's tree representation [16], Parrow's Interaction Diagrams [15], and Milner's $\pi$-nets [12].

We would also like to face the intractability of the problem of deciding equivalence for more general $\pi$-calculus agents (without finite control). The inequality problem is semidecidable — and indeed due to the "on-the-fly" implementation of our algorithm we can provide such results — but the equality problem requires a tool running in some sort of semi-automatic mode, asking the human user for assistance at different points, e.g. for choosing strategy and tactics to solve a given problem.

## Acknowledgement

We would like to thank Davide Sangiorgi for initiating the theoretical developments explored in this paper, and for providing us with numerous comments and corrections. Joachim Parrow and Lars-åke Fredlund also contributed useful comments on early drafts of this paper, and Lars-åke provided model code from which the first author learned a great deal of Standard ML.

# References


[1] N.G. de Bruijn. Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indagationes Mathematicae*, 34:381–392. North-Holland, 1972.

[2] M. Dam. Model Checking Mobile Processes. In Proceedings of CONCUR'93. *Lecture Notes in Computer Science* 715. E. Best (ed). pp22–36. Springer-Verlag, 1993.

[3] J-C. Fernandez. Aldébaran: A tool for verification of communicating processes. Technical Report RTC 14, IMAG, Grenoble, 1989.

[4] J-C. Fernandez and L. Mounier. "On-the-fly" verification of behavioural equivalences and preorders. In Proceedings of CAV'91. 1991.

[5] L.-å. Fredlund and F. Orava. Modelling Dynamic Communication Structures in LOTOS. In Proceedings of FORTE'91, K.R. Parker and G.A. Rose (eds). pp185–200. North-Holland, 1992.

[6] H. Garavel and J. Sifakis. Compilation and verification of LOTOS specifications. In Proceedings of Protocol Specification, Testing, and Verification X, 1990.

[7] S.J. Gay. A Sort Inference Algorithm for the Polyadic $\pi$-Calculus. In Proceedings of 20th ACM Symp. on Principles of Programming Languages, ACM Press, 1993.

[8] M. Hennessy and H. Lin. Symbolic bisimulations. Research Report TR1/92. University of Sussex, 1992.

[9] E. Madelaine. Verification tools from the CONCUR project. *Bulletin of the European Association of Theoretical Computer Science* 47, pp110–126, June 1992.

[10] R. Milner. **Communication and Concurrency**. Prentice-Hall, 1989.

[11] R. Milner. The polyadic $\pi$-calculus: a tutorial. Research Report ECS-LFCS-91-180. University of Edinburgh, October 1991.

[12] R. Milner. Action Structures for the $\pi$-Calculus. Research Report ECS-LFCS-93-264. University of Edinburgh, May 1993.

[13] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes (Parts I and II). *Journal of Information and Computation*, 100:1–77, September 1992.

[14] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4:497–543, 1992.

[15] J. Parrow. Interaction Diagrams. Swedish Institute of Computer Science Research Report R93:06, 1993. (To appear in Proceedings of REX'93, Springer-Verlag.)

[16] D. Sangiorgi. A theory of bisimulation for the $\pi$-calculus. In Proceedings of CONCUR'93. *Lecture Notes in Computer Science* 715. E. Best (ed). pp127–142. Springer-Verlag, 1993.

[17] P.H.J. van Eijk, C.A. Vissers and M. Diaz (eds). **The Formal Description Technique LOTOS**. North-Holland, 1989.

[18] B. Victor. Forthcoming licentiate thesis, Uppsala University, 1994.