

Excommunication: Transforming π -Calculus Specifications to Remove Internal Communication

G.W. Hamilton*, B. Aziz*, D. Gray*, J. Power†, D. Sinclair*

*School of Computer Applications, Dublin City University, Glasnevin, Dublin 9, Ireland

†Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland

October 8, 2001

Abstract

In this paper, we present a new automatic transformation algorithm which can automatically transform π -calculus processes to remove all explicit internal communication. We prove that the transformation preserves full bisimulation equivalence, and also that the transformation always terminates.

1 Introduction

The π -calculus [?] is a calculus of mobile processes, in which mobility is modelled by the movement of links between processes. π -calculus specifications tend to be written in a style which involves intermediate internal communication. As a simple example, consider the following π -calculus specification:

$$(\nu i)(B[l, i] | C[i, r])$$

where

$$B \stackrel{\text{def}}{=} (l, i).l(x).\bar{i}\langle x \rangle.B[l, i]$$
$$C \stackrel{\text{def}}{=} (i, r).i(x).\bar{r}\langle x \rangle.C[i, r]$$

This is the definition of two single cell buffers which are chained together. One cell receives its input at l , and emits this at i . The other cell receives its input at i , and emits this at r . This specification therefore contains some unnecessary internal communication. This can be transformed instead to the following specification from which this internal communication has been removed:

$$\begin{aligned}
& D[l, r] \\
\text{where} \\
D & \stackrel{\text{def}}{=} (l, r).l(x).E[l, r, x] \\
E & \stackrel{\text{def}}{=} (l, r, x).\bar{r}\langle x \rangle.D[l, r] + l(y).F[l, r, x, y] \\
F & \stackrel{\text{def}}{=} (l, r, x, y).\bar{r}\langle x \rangle.E[l, r, y]
\end{aligned}$$

This new specification is full bisimilar to the original one.

Automatic transformation techniques have been widely studied for many programming paradigms (see [?] for an overview), but few similar techniques have been developed for the π -calculus (one notable exception can be found in [?]). In this paper, we present a transformation algorithm called *excommunication* which removes all explicit internal communication from any given π -calculus specification, thus being able to perform the transformation above. We prove that the transformation preserves full bisimulation equivalence, and also that the transformation always terminates.

The remainder of this paper is structured as follows: In Section 2, we define the syntax and semantics of the π -calculus. In Section 3, we define the excommunication algorithm, and in Section 4 we prove the excommunication theorem which states that excommunication preserves weak bisimulation equivalence and always terminates. Section 5 concludes.

2 The π -Calculus

In this section, we describe the syntax and semantics of the π -calculus.

Definition 2.1 (Syntax of the π -Calculus) The syntax of the π -calculus is shown in Fig. 1. \square

A specification consists of a process and a set of named process definitions. Note that we do not include the replication operator (!), as recursion can be represented using named processes. Within inputs $x(y).P$ and restrictions $(\nu y)P$, the name y is bound within P . A name is free in a process if it is not bound. We denote the set of names which are free in process P by $fn(P)$.

Definition 2.2 (Structural Congruence) The structural congruence rules of the π -calculus are defined as shown in Figure 2. \square

Definition 2.3 (Reduction) The reduction rules of the π -calculus are defined as shown in Figure 3. \square

$S ::= P$	Specification
where	
$p_1 \stackrel{\text{def}}{=} (x_{11} \dots x_{1k_1}).P_1$	
\vdots	Process Definitions
$p_n \stackrel{\text{def}}{=} (x_{n1} \dots x_{nk_n}).P_n$	
$P ::= \mathbf{0}$	Null Process
$x(y).P$	Input Prefix
$\bar{x}(y).P$	Output Prefix
$\tau.P$	Silent Action Prefix
$[x = y]P$	Match Prefix
$(\nu x)P$	Restriction
$P_1 + P_2$	Sum
$p[x_1 \dots x_n]$	Defined Process Application
$P_1 P_2$	Parallel Composition

Figure 1: Syntax of the π -Calculus

We now define full bisimilarity, the congruence relation which is used to prove the transformations in this paper correct. In the following, α ranges over all actions, and β ranges over all actions except the silent action τ .

Notation 2.4 (Transition Relations) The transition relation $P \xrightarrow{\alpha} P'$ means that process P can evolve into P' by performing action α .

Notation 2.5 (Substitution) A substitution $\sigma = \{y_1, \dots, y_n / x_1, \dots, x_n\}$ is a function on names such that:

$$x\sigma = \begin{cases} y_i, & \text{if } x = x_i \\ x, & \text{otherwise} \end{cases}$$

Definition 2.6 (Application of Substitution) The process $P\sigma$ obtained by applying σ to P is defined as follows:

$[x = x]P$	\equiv	P	SC-MAT
$P_1 + (P_2 + P_3)$	\equiv	$(P_1 + P_2) + P_3$	SC-SUM-ASSOC
$P_1 + P_2$	\equiv	$P_2 + P_1$	SC-SUM-COMM
$P + \mathbf{0}$	\equiv	P	SC-SUM-INACT
$P_1 (P_2 P_3)$	\equiv	$(P_1 P_2) P_3$	SC-COMP-ASSOC
$P_1 P_2$	\equiv	$P_2 P_1$	SC-COMP-COMM
$P \mathbf{0}$	\equiv	P	SC-COMP-INACT
$(\nu x)(\nu y)P$	\equiv	$(\nu y)(\nu x)P$	SC-RES
$(\nu x)\mathbf{0}$	\equiv	$\mathbf{0}$	SC-RES-INACT
$(\nu x)(P_1 P_2)$	\equiv	$P_1 (\nu x)P_2$, if $x \notin fn(P_1)$	SC-RES-COMP
$p[x_1 \dots x_n]$	\equiv	$P[x_1/y_1 \dots x_n/y_n]$	SC-APP
		where $p \stackrel{\text{def}}{=} (y_1 \dots y_n).P$	

Figure 2: Structural Congruence Rules of the π -Calculus

$\mathbf{0}\sigma$	$\stackrel{\text{def}}{=}$	$\mathbf{0}$
$(x(y).P)\sigma$	$\stackrel{\text{def}}{=}$	$(x\sigma)(y).(P\sigma)$
$(\bar{x}\langle y \rangle).P\sigma$	$\stackrel{\text{def}}{=}$	$\bar{x}\sigma\langle y\sigma \rangle.(P\sigma)$
$(\tau.P)\sigma$	$\stackrel{\text{def}}{=}$	$\tau.(P\sigma)$
$([x = y]P)\sigma$	$\stackrel{\text{def}}{=}$	$[(x\sigma) = (y\sigma)](P\sigma)$
$(\nu x)P\sigma$	$\stackrel{\text{def}}{=}$	$(\nu x)(P\sigma)$
$(P_1 + P_2)\sigma$	$\stackrel{\text{def}}{=}$	$(P_1\sigma) + (P_2\sigma)$
$(p[x_1 \dots x_n])\sigma$	$\stackrel{\text{def}}{=}$	$(p[(x_1\sigma) \dots (x_n\sigma)])$
$(P_1 P_2)\sigma$	$\stackrel{\text{def}}{=}$	$(P_1\sigma) (P_2\sigma)$

It is assumed here that the bound names of the process are distinct from the free names, and from the names of the substitution.

Notation 2.7 (Weak Transition Relations) The weak transition relation $\overset{\alpha}{\Rightarrow}$ is defined as $\overset{\alpha}{\Rightarrow} \stackrel{\text{def}}{=} \Longrightarrow \overset{\alpha}{\Rightarrow}$, where \Longrightarrow is the reflexive transitive closure of $\overset{\tau}{\rightarrow}$.

Definition 2.8 (Bisimilarity) Bisimilarity is the largest symmetric relation, \approx , such that whenever $P \approx Q$:

1. $P \overset{\beta}{\rightarrow} P'$ implies $Q \overset{\beta}{\Rightarrow} \approx P'$

$$\begin{array}{c}
\frac{}{(\bar{x}\langle y \rangle.P_1 + P_2)|(x(z).P_3 + P_4) \longrightarrow P_1|P_3\{y/z\}} \text{ R-INTER} \\
\\
\frac{}{\tau.P_1 + P_2 \longrightarrow P_1} \text{ R-TAU} \\
\\
\frac{P_1 \longrightarrow P'_1}{P_1|P_2 \longrightarrow P'_1|P_2} \text{ R-PAR} \\
\\
\frac{P \longrightarrow P'}{(\nu x)P \longrightarrow (\nu x)P'} \text{ R-RES} \\
\\
\frac{P_1 \equiv P_2 \longrightarrow P'_2 \equiv P'_1}{P_1 \longrightarrow P'_1} \text{ R-STRUCT}
\end{array}$$

Figure 3: Reduction Rules of the π -Calculus

2. $P \xrightarrow{\tau} P'$ implies $Q \Longrightarrow \approx P'$

□

Definition 2.9 (Full Bisimilarity) Processes P and Q are full bisimilar, $P \approx^C Q$, if $P\sigma \approx Q\sigma$ for every substitution σ . □

We now define the reduced form of a process which contains no explicit parallel composition, and hence no explicit internal communication.

Definition 2.10 (Reduced Form) Reduced form is defined as shown in Fig. 4. □

$$\begin{array}{l}
S' ::= P' \textbf{ where } p_1 \stackrel{\text{def}}{=} (x_{11} \dots x_{1k_1}).P'_1 \dots p_n \stackrel{\text{def}}{=} (x_{n1} \dots x_{nk_n}).P'_n \\
P' ::= \mathbf{0} \mid x(y).P' \mid \bar{x}\langle y \rangle.P' \mid \tau.P' \mid [x = y]P' \mid (\nu x)P \mid P'_1 + P'_2 \mid p[x_1 \dots x_n]
\end{array}$$

Figure 4: Reduced Form

3 The Excommunication Algorithm

In this section, we present the excommunication algorithm. This is a set of transformation rules which convert a given specification into an equivalent specification from which all explicit internal communication has been removed.

Definition 3.1 (Excommunication Algorithm) The transformation rules for the excommunication algorithm are shown in Figs. 5 and 6. \square

The additional parameter ρ within these rules is the set of names which are internal to a process, and can therefore be removed by these rules. These are the names which are generated within restrictions, and are output over internal names only.

Rules $(\mathcal{T}1)$ - $(\mathcal{T}9)$ cover all possible kinds of process (null, prefixed by input, output or silent action, matching, restriction, sum, user-defined process and parallel composition). In rule $(\mathcal{T}1)$, the result of transforming a null process is a null process. In rule $(\mathcal{T}2)$, an input action is extracted out of the process being transformed if it is not over an internal name. Otherwise, the result is the null process. In rule $(\mathcal{T}3)$, an output action is extracted out of the process being transformed if it is not over an internal name. If the name which is being output is internal, then scope extrusion will take place, so the name being output is restricted within the surrounding context and this name is removed from the set of internal names. Otherwise, the result is the null process. In rule $(\mathcal{T}4)$, a silent action prefix is removed. In rule $(\mathcal{T}5)$, matching is performed if the two names are the same. If either of the two names being matched is internal, then the matching fails and the result is the null process; otherwise the matching operation remains. In rule $(\mathcal{T}6)$, a restriction is removed from the process being transformed, and the restricted name is added to the set of internal names. In rule $(\mathcal{T}7)$ for a sum, the summands are transformed separately, and the result is the sum of the two transformed processes. In rule $(\mathcal{T}8)$, a user-defined process call is replaced by the process body, with the formal names of the body replaced by the actual names in the call. In rule $(\mathcal{T}9)$ for a parallel composition, the composed processes are transformed separately, and the result is obtained by transforming the different possible compositions of these transformed processes. Note that when each of the processes are transformed separately, the initial set of internal names supplied to each process is empty. This is because internal names which are shared between the two processes should not be removed at this stage; they are necessary to allow for communication between the two processes. Internal names which remain after transforma-

tion of the two processes will be removed by the application of \mathcal{P} to the results of this transformation.

Rules $(\mathcal{P}1)$ - $(\mathcal{P}8)$ cover all possible kinds of process for the first process argument (as both processes have been transformed by the function \mathcal{T} , it is not possible for either of them to be a parallel composition). In rule $(\mathcal{P}1)$, if the first process is null, then the result is the null process. In rule $(\mathcal{P}2)$, if the first process is prefixed by an input action, then communication may or may not take place with the second process. The result of the transformation in this case is therefore the sum of these two possibilities. The process P_3 will result in the case where communication does take place. In this case, if the second process is prefixed by an output action, then the residues of the two processes are composed only if the names in the two actions are the same; this is therefore guarded by a matching operation (this is actually necessary to ensure that the transformation preserves full bisimulation equivalence). If the second process is not prefixed by an output action, then no communication can take place, so the result is the null process. The process P_4 will result in the case where communication does not take place. In this case, the input action is pulled outside the different possible compositions of the residue of the first process with the second process, if it is not over an internal name (note that this is valid only if there is no clash between the bound name and the free names in the second process; alpha conversion may need to be applied to ensure that this is the case). Otherwise the result is the null process. In rule $(\mathcal{P}3)$, if the first process is prefixed by an output action, then again communication may or may not take place with the second process. The result of the transformation in this case is therefore the sum of these two possibilities. The process P_3 will result in the case where communication does take place. In this case, if the second process is prefixed by an input action, then the residues of the two processes are composed only if the names in the two actions are the same; this is therefore guarded by a matching operation (this is actually necessary to ensure that the transformation preserves full bisimulation equivalence). If the second process is not prefixed by an input action, then no communication can take place, so the result is the null process. The process P_4 will result in the case where communication does not take place. In this case, the output action is pulled outside the different possible compositions of the residue of the first process with the second process if it is not over an internal name. If the name which is being output is internal, then scope extrusion will take place, so the name being output is restricted within the surrounding context and this name is removed from the set of internal names. Otherwise, the result is the null process. In rule $(\mathcal{P}4)$, if the first process is prefixed by a

silent action, then the silent action is removed and the result is obtained by transforming the composition of the residue of the first process with the second process. In rule ($\mathcal{P}5$), if the first process is prefixed by matching, matching is performed if the two names are the same, and the result is obtained by transforming the composition of the residue of the first process with the second process. If either of the two names being matched is internal, then the matching fails and the result is the null process; otherwise the matching operation is pulled outside the process obtained by transforming the composition of the residue of the first process with the second process (this is valid only if there no clash between the names in the match and the free names in the second process, so alpha conversion may need to be applied to ensure that this is the case). In rule ($\mathcal{P}6$), if the first process is a restriction, then the restriction is removed and the result is obtained by transforming the composition of the residue of the first process with the second process, with the restricted name added to the set of internal names. This rule is valid only if there no clash between the restricted name and the free names in the second process, so alpha conversion may need to be applied to ensure that this is the case. In rule ($\mathcal{P}7$), if the first process is a sum, then each of the summands are composed with the second process, and these compositions are then further transformed to produce another sum. In rule ($\mathcal{P}8$), if the first process is a user-defined process call, then this is replaced by the process body, with the formal names of the body replaced by the actual names in the call. The result is obtained by transforming the composition of this unfolding with the second process.

As defined so far, the excommunication algorithm will not necessarily terminate. Termination is achieved only through the introduction of appropriate new process definitions. Any infinite sequence of transformation steps must involve the unfolding of a user-defined process call. A new process definition is therefore introduced before the application of rules ($\mathcal{T}8$) and ($\mathcal{P}8$). The arguments of the process definition are the names within the process which was about to be transformed which are not internal (i.e. are not contained in ρ). The right hand side of the process definition is the process which was about to be transformed. When a process is encountered later in the transformation which matches the right hand side of one of these process definitions (modulo renaming of variables), it is replaced by an appropriate call of the corresponding process definition. This folding is defined more formally as follows.

Definition 3.2 (Folding) Transformation rules ($\mathcal{T}8$) and ($\mathcal{P}8$) must be changed as shown in Fig. 7 to define folding explicitly. \square

The additional parameter ϕ contains the set of processes which have been encountered before during the transformation and the associated process name which was introduced. This additional parameter must also be passed to all other transformation rules.

4 The Excommunication Theorem

The excommunication theorem can now be stated as follows.

Theorem 4.1 (Excommunication Theorem) Every Π -calculus specification can be transformed by the excommunication algorithm into an equivalent specification which is in reduced form. \square

In order to prove that the result of the transformation is equivalent to the original specification, we prove by structural induction that the transformation rules preserve full bisimulation equivalence.

Lemma 4.2 (On Equivalence) $\forall P, Q \in \text{Spec} : \mathcal{T}[[P]] = Q \Rightarrow P \approx^C Q$ \square

Proof

The proof is by structural induction on the rules \mathcal{T} and \mathcal{P} .

\square

In order to prove that the algorithm always terminates, we show that there is a bound on the size of processes which are encountered during transformation. First of all, it must be defined what is meant by the size of a process.

Definition 4.3 (Size of Processes) The size of a process is defined as shown in Fig. 8. \square

Lemma 4.4 (On Size of Processes) If the size of all process definitions is bounded by s , and the original process to be transformed is also bounded by s , then the size of all processes encountered by the excommunication algorithm is bounded by s .

$$\mathcal{T}[[P]] \rho = \dots (\mathcal{T}[[P']] \rho) \dots \wedge \mathcal{S}[[P]] \leq s \Rightarrow \mathcal{S}[[P']] \leq s$$

$$\mathcal{P}[[P_1]][[P_2]] \rho = \dots (\mathcal{P}[[P'_1]][[P'_2]] \rho) \dots \wedge \mathcal{S}[[P_1]] \leq s \wedge \mathcal{S}[[P_2]] \leq s \Rightarrow \mathcal{S}[[P'_1]] \leq s \wedge \mathcal{S}[[P'_2]] \leq s$$

\square

Proof

The proof is by structural induction on the rules \mathcal{T} and \mathcal{P} .

□

5 Conclusion

References

$$\begin{array}{ll}
(\mathcal{T}1) & \mathcal{T}[\mathbf{0}] \rho = \mathbf{0} \\
(\mathcal{T}2) & \mathcal{T}[x(y).P] \rho = \begin{cases} \mathbf{0}, & \text{if } x \in \rho \\ x(y).(\mathcal{T}[P] \rho), & \text{otherwise} \end{cases} \\
(\mathcal{T}3) & \mathcal{T}[\bar{x}(y).P] \rho = \begin{cases} \mathbf{0}, & \text{if } x \in \rho \\ (\nu y)\bar{x}(y).(\mathcal{T}[P] (\rho \setminus \{y\})), & \text{if } y \in \rho \\ \bar{x}(y).(\mathcal{T}[P] \rho), & \text{otherwise} \end{cases} \\
(\mathcal{T}4) & \mathcal{T}[\tau.P] \rho = \mathcal{T}[P] \rho \\
(\mathcal{T}5) & \mathcal{T}[[x = y]P] \rho = \begin{cases} \mathcal{T}[P] \rho, & \text{if } x = y \\ \mathbf{0}, & \text{if } x \in \rho \vee y \in \rho \\ [x = y](\mathcal{T}[P] \rho), & \text{otherwise} \end{cases} \\
(\mathcal{T}6) & \mathcal{T}[(\nu x)P] \rho = \mathcal{T}[P] (\rho \cup \{x\}) \\
(\mathcal{T}7) & \mathcal{T}[P_1 + P_2] \rho = (\mathcal{T}[P_1] \rho) + (\mathcal{T}[P_2] \rho) \\
(\mathcal{T}8) & \mathcal{T}[p[x_1 \dots x_n]] \rho = \mathcal{T}[P[x_1/y_1 \dots x_n/y_n]] \rho \\
& \text{where} \\
& p \stackrel{\text{def}}{=} (y_1 \dots y_n).P \\
(\mathcal{T}9) & \mathcal{T}[P_1 | P_2] \rho = (\mathcal{P}[P_3][P_4] \rho) + (\mathcal{P}[P_4][P_3] \rho) \\
& \text{where} \\
& P_3 = \mathcal{T}[P_1] \{\} \\
& P_4 = \mathcal{T}[P_2] \{\}
\end{array}$$

Figure 5: Transformation Rules \mathcal{T} for Excommunication

$$\begin{aligned}
(\mathcal{P}1) \quad & \mathcal{P}[\mathbf{0}][P] \rho = \mathbf{0} \\
(\mathcal{P}2) \quad & \mathcal{P}[x(y).P_1][P_2] \rho = P_3 + P_4 \\
& \text{where} \\
& P_3 = \begin{cases} \mathcal{P}[[x = x']P_1[z/y]][P_2'] \rho, & \text{if } P_2 = \bar{x}'\langle z \rangle.P_2' \\ \mathbf{0}, & \text{otherwise} \end{cases} \\
& P_4 = \begin{cases} \mathbf{0}, & \text{if } x \in \rho \\ (x(y).\mathcal{P}[P_1][P_2]) \rho + (x(y).\mathcal{P}[P_2][P_1]) \rho, & \text{otherwise} \end{cases} \\
(\mathcal{P}3) \quad & \mathcal{P}[\bar{x}\langle y \rangle.P_1][P_2] \rho = P_3 + P_4 \\
& \text{where} \\
& P_3 = \begin{cases} \mathcal{P}[[x = x']P_1][P_2'[y/z]] \rho, & \text{if } P_2 = x'(z).P_2' \\ \mathbf{0}, & \text{otherwise} \end{cases} \\
& P_4 = \begin{cases} \mathbf{0}, & \text{if } x \in \rho \\ ((\nu y)\bar{x}\langle y \rangle.\mathcal{P}[P_1][P_2] (\rho \setminus \{y\})) + \\ ((\nu y)\bar{x}\langle y \rangle.\mathcal{P}[P_2][P_1] (\rho \setminus \{y\})), & \text{if } y \in \rho \\ (\bar{x}\langle y \rangle.\mathcal{P}[P_1][P_2]) \rho + (\bar{x}\langle y \rangle.\mathcal{P}[P_2][P_1]) \rho, & \text{otherwise} \end{cases} \\
(\mathcal{P}4) \quad & \mathcal{P}[\tau.P_1][P_2] \rho = \mathcal{P}[P_1][P_2] \rho \\
(\mathcal{P}5) \quad & \mathcal{P}[[x = y]P_1][P_2] \rho = \\
& \begin{cases} \mathcal{P}[P_1][P_2] \rho, & \text{if } x = y \\ \mathbf{0}, & \text{if } x \in \rho \vee y \in \rho \\ [x = y]\mathcal{P}[P_1][P_2] \rho, & \text{otherwise} \end{cases} \\
(\mathcal{P}6) \quad & \mathcal{P}[(\nu x)P_1][P_2] \rho = \mathcal{P}[P_1][P_2] (\rho \cup \{x\}) \\
(\mathcal{P}7) \quad & \mathcal{P}[P_1 + P_2][P_3] \rho = (\mathcal{P}[P_1][P_3] \rho) + (\mathcal{P}[P_2][P_3] \rho) \\
(\mathcal{P}8) \quad & \mathcal{P}[p[x_1 \dots x_n]][P_2] \rho = \mathcal{P}[P_1[x_1/y_1 \dots x_n/y_n]][P_2] \rho \\
& \text{where} \\
& p \stackrel{\text{def}}{=} (y_1 \dots y_n).P_1
\end{aligned}$$

Figure 6: Transformation Rules \mathcal{P} for Excommunication

$$\begin{aligned}
(\mathcal{T}8) \quad & \mathcal{T}[[p[x_1 \dots x_n]]] \rho \phi = \\
& \begin{cases} p'[x''_1 \dots x''_k], & \text{if } (p' \stackrel{\text{def}}{=} (y''_1 \dots y''_k).P') \in \phi \\ & \text{and } (y''_1 \dots y''_k).P' = \\ & ((x''_1 \dots x''_k).p[x_1 \dots x_n])[y'_1/x'_1 \dots y'_j/x'_j] \\ p''[x''_1 \dots x''_k], & \text{otherwise} \end{cases} \\
& \text{where} \\
& \{x'_1 \dots x'_j\} = fn(p[x_1 \dots x_n]) \\
& \{x''_1 \dots x''_k\} = fn(p[x_1 \dots x_n]) \setminus \rho \\
& \phi' = \phi \cup (p'' \stackrel{\text{def}}{=} (x''_1 \dots x''_k).p[x_1 \dots x_n]) \\
& p \stackrel{\text{def}}{=} (z_1 \dots z_n).P \\
& p'' \stackrel{\text{def}}{=} (x''_1 \dots x''_k).\mathcal{T}[[P[x_1/z_1 \dots x_n/z_n]]] \rho \phi' \\
\\
(\mathcal{P}8) \quad & \mathcal{P}[[p[x_1 \dots x_n]]][P_2] \rho \phi = \\
& \begin{cases} p'[x''_1 \dots x''_k], & \text{if } (p' \stackrel{\text{def}}{=} (y''_1 \dots y''_k).P') \in \phi \\ & \text{and } (y''_1 \dots y''_k).P' = \\ & ((x''_1 \dots x''_k).p[x_1 \dots x_n]|P_2)[y'_1/x'_1 \dots y'_j/x'_j] \\ p''[x''_1 \dots x''_k], & \text{otherwise} \end{cases} \\
& \text{where} \\
& \{x'_1 \dots x'_j\} = fn(p[x_1 \dots x_n]|P_2) \\
& \{x''_1 \dots x''_k\} = fn(p[x_1 \dots x_n]|P_2) \setminus \rho \\
& \phi' = \phi \cup (p'' \stackrel{\text{def}}{=} (x''_1 \dots x''_k).(p[x_1 \dots x_n]|P_2)) \\
& p \stackrel{\text{def}}{=} (z_1 \dots z_n).P_1 \\
& p'' \stackrel{\text{def}}{=} (x''_1 \dots x''_k).\mathcal{P}[[P_1[x_1/z_1 \dots x_n/z_n]]][P_2] \rho \phi'
\end{aligned}$$

Figure 7: Folding Rules for Excommunication Algorithm

$$\begin{aligned}
\mathcal{S}[\mathbf{0}] &= 1 \\
\mathcal{S}[x(y).P] &= 1 + \mathcal{S}[P] \\
\mathcal{S}[\bar{x}(y).P] &= 1 + \mathcal{S}[P] \\
\mathcal{S}[(\nu x)P] &= 1 + \mathcal{S}[P] \\
\mathcal{S}[P_1 + P_2] &= \mathcal{S}[P_1] + \mathcal{S}[P_2] \\
\mathcal{S}[p[x_1 \dots x_n]] &= n \\
\mathcal{S}[P_1 | P_2] &= \mathcal{S}[P_1] + \mathcal{S}[P_2]
\end{aligned}$$

Figure 8: Size of Processes