Characterizing Bisimulation Congruence in the π–Calculus*

Xinxin Liu

School of Cognitive and Computing Sciences

University of Sussex

Brighton BN1 9QH

England

Janurary 1994

Abstract

This paper presents a new characterization of the bisimulation congruence and D–bisimulation equivalences of the π–calculus. The characterization supports a bisimulation–like proof technique which avoids explicit case analysis by taking a dynamic point of view of actions a process may perform, thus providing a new way of proving bisimulation congruence. The semantic theory of the π–calculus is presented here without the notion of α–equivalence.

1 Motivation

The π–calculus, introduced in [MPW92a], presents a model of concurrent computation based upon the notion of *naming*. It can be seen as an extension of the theory of CCS [Mil89] (and other similar process algebras) in that names (references) are the subject of communication. This introduces mobility into process algebras. Such an extension allows us to clearly express many fundamental programming features which could at best be described indirectly in CCS.

The theory of CCS has been quite successful for specifying and verifying concurrent systems. The success is due to a solid equality theory based on the notion of bisimulation [Par81, Mil89]. Bisimulation has many nice properties. It induces a congruence relation for CCS constructions, thus supporting compositionality. It admits a very pleasant proof technique based on fixed point induction [Par81]. The proof technique not only provides a means of establishing the equality theory but also opens up a direct way of program verification.

*This work has been supported by the SERC grant GR/H16537

Corresponding to the bisimulation equivalence in CCS, there are two main equalities in the π–calculus: *ground bisimulation equivalence* and *bisimulation congruence*. The notion of ground bisimulation is a natural generalization of that

and y, we can make a step forward by reducing the problem of proving $P \sim Q$ into two subproblems of proving $P \sim_{\{x,y\}} Q$ and $P\{y/x\} \sim Q\{y/x\}$.

As a simple example let us prove the following

$$x|\bar{y} + x|\bar{x} \sim x.\bar{y} + \bar{y}.x + x|\bar{x}$$

According to the above discussion, this can be reduced to proving both of the following

$$x|\bar{y} + x|\bar{x} \sim_{\{x,y\}} x.\bar{y} + \bar{y}.x + x|\bar{x}$$

and

$$x|\bar{x} + x|\bar{x} \sim x.\bar{x} + \bar{x}.x + x|\bar{x}$$

Now in the first equivalence, the distinction $\{x, y\}$ requires that any pairs of different free names of the processes must be distinct under substitution. So the first equivalence is guaranteed by $x|\bar{y} + x|\bar{x} \dot{\sim} x.\bar{y} + \bar{y}.x + x|\bar{x}$. A similar argument concludes that the second equivalence is guaranteed by $x|\bar{x} + x|\bar{x} \dot{\sim} x.\bar{x} + \bar{x}.x + x|\bar{x}$, and now we can use the proof technique for $\dot{\sim}$. In the general case, we may have to further reduce the subproblems before $\dot{\sim}$ would guarantee D–bisimilarity of two processes.

In fact here we are doing case analysis to divide the substitution set into smaller subsets until $\dot{\sim}$ can solve the subproblems. This cannot be a satisfactory proof strategy in the general case. Because on many occasions, in order to conclude congruence between processes, it is unnecessary to do case analysis until ground bisimulation can come into play. The most obvious example is to prove $P \sim P$ for whatever complicated P. The aim of the paper is to introduce a proof technique which allows us to do the necessary case analysis implicitly and to keep it to minimum. A key point is to adopt a dynamic point of view of names by introducing conditional commitment. The approach here is inspired by the work of *symbolic bisimulation* of [HL92], where the idea of dividing a value space is used in a proof technique to establish bisimulation equivalence between value passing processes. In fact this paper applies the main idea in [HL92] while taking advantage of the primitive structure of the π–calculus.

As another contribution, this paper demonstrates that the core of the theory of π–calculus can be presented without using the notion of α–equivalence. Traditionally, α–equivalence, or "syntactic identity modulo α–conversion", plays a very important role in theories in which variables can be bound. However it is also well known that proofs involving this notion are very tedious. A desirable approach is that the basic semantic results are worked out independent of this notion. Of couse, afterwards the semantics

that of $\nu x[x]P$ is a bound name x. These are the two standard or normal forms of concretion. Later we will see that any concretion can be *normalized* to one of these forms. A negative prefix $\bar{x}.C$ now corresponds to the output prefix in CCS, which outputs the name part of C at port x and then continue with the process part of C. In its standard form, an abstraction $(x)P$ is in fact a function which for each name y gives a process $P\{y/x\}$, where we use the notation $P\{y/x\}$ to describe the syntactic substitution of y for all free occurrences of x in P. Thus free occurrences of x in P are bound by (x). Also we will see later that any abstraction can be normalized to such a form. Now a positive prefix $x.(y)P$ corresponds to the input prefix in CCS, which receives a name z at port x and then behaves like $P\{z/y\}$. Finally the replication $!P$ means $P|P|\ldots$ which plays the role of recursion. The pure synchronization structures $x.P, \bar{x}.P$ are not essential. They are included for the benefit of writing simple expressions in examples. Moreover $\alpha.\mathbf{0}$ is often abbreviated to α (we have already used this abbreviation in the earlier examples).

To summarize whether a name is free or bound in an expression A, we give the following definition.

Definition 2.1 *We write* $\mathtt{fn}(A)$ *for the set of free names of* A *— names which are neither bound by abstraction nor by restriction.* $\mathtt{fn}(A)$ *is defined inductively on the structure of* A*:*

$$
\begin{aligned}
&\mathtt{fn}(\mathbf{0}) = \emptyset,\\
&\mathtt{fn}(\tau.P) = \mathtt{fn}(!P) = \mathtt{fn}(P)\\
&\mathtt{fn}(P + P') = \mathtt{fn}(P) \cup \mathtt{fn}(P')\\
&\mathtt{fn}(P|A) = \mathtt{fn}(A|P) = \mathtt{fn}(P) \cup \mathtt{fn}(A)\\
&\mathtt{fn}(x.A) = \mathtt{fn}(\bar{x}.A) = \{x\} \cup \mathtt{fn}(A)\\
&\mathtt{fn}([x]P) = \{x\} \cup \mathtt{fn}(P)\\
&\mathtt{fn}((x)P) = \mathtt{fn}(P) - \{x\}\\
&\mathtt{fn}(\nu x A) = \mathtt{fn}(A) - \{x\}
\end{aligned}
$$

Sometimes we will write $\mathtt{fn}(A, B)$ *as an abbreviation for* $\mathtt{fn}(A) \cup \mathtt{fn}(B)$.

2.2 Simultaneous Substitution

We have informally used notation $P\{y/x\}$ for the substitution of y for all free occurrences of y in P. It need to be clarified what this substitution exactly means, as there are possiblely bound names in a term, we have to be careful to avoid name clash when making such substitutions. In this paper we take the approach of *simultaneous substitution* introduced by Alan Stoughton [Sto88] for the lambda calculus. The definition below is a specialized version of that in [Sto88] in the sense that a name may only be substituted by another name in the π–calculus while a variable may be substituted by a term in the lambda calculus.

We assume that there is a function `fresh` which for a given set N of names will produce a name $\mathtt{fresh}(N)$ such that $\mathtt{fresh}(N) \notin N$ (or we can assume some

proper ordering on $\mathcal{N}$ and take $\texttt{fresh}(N)$ to be the smallest one which is not in $\mathcal{N}$). Now, substitution and its application to agents is formally given by the following definition.

Definition 2.2 *A substitution is a function from $\mathcal{N}$ to $\mathcal{N}$. We use σ, ρ to range over substitutions, and postfix substitutions in application. For a given substitution σ, $\sigma\{y_i/x_i\}_{1\leq i\leq n}$ denotes the following updated substitution of σ:*

$$x\sigma\{y_i/x_i\}_{1\leq i\leq n} = \begin{cases} y_i & \text{if } x = x_i \text{ for } 1 \leq i \leq n \\ x\sigma & \text{otherwise} \end{cases}$$

For two substitutions σ, ρ, we write $\sigma\circ\rho$ for the composition *of σ with ρ, which is the substitution such that for $x \in \mathcal{N}$, $x\sigma\circ\rho = (x\sigma)\rho$ (note since we postfix substitution in application, the order of composition is the reverse of that in normal function composition).*

For any substitution σ and name x, as a convention we let $\bar{x}\sigma = \overline{x\sigma}$ and $\tau\sigma = \tau$, thus extending the domain and range of substitutions to the set of all actions. We write $A\sigma$ for the agent obtained by applying the substitution σ to agent A. It is defined on the structure of A:

$$\begin{array}{lcll}
\mathbf{0}\sigma & \equiv & \mathbf{0} & \\
(\alpha.A)\sigma & \equiv & \alpha\sigma.A\sigma & \\
(P+P')\sigma & \equiv & P\sigma + P'\sigma & \\
(P|A)\sigma & \equiv & P\sigma|A\sigma & \\
(A|P)\sigma & \equiv & A\sigma|P\sigma & \\
(!P)\sigma & \equiv & !P\sigma & \\
(\nu x A)\sigma & \equiv & \nu z A\sigma\{z/x\} & z = \texttt{fresh}(\{y\sigma \mid y \in \texttt{fn}(A) - \{x\}\}) \\
((x)P)\sigma & \equiv & (z)P\sigma\{z/x\} & z = \texttt{fresh}(\{y\sigma \mid y \in \texttt{fn}(P) - \{x\}\}) \\
([x]P)\sigma & \equiv & [x\sigma]P\sigma &
\end{array}$$

In the above, z is chosen to avoid name clash by using `fresh`.

As demonstrated in [Sto88], this simultaneous substitution is easier to work with than standard single substitution. The effect of single substitution of x for y in P is now obtained by applying simultaneous substitution $\iota\{x/y\}$ on P, where ι is the identity map. So, from now on, we will write $P\iota\{x/y\}$ for single substitution instead of $P\{x/y\}$. The following conventions are adopted to avoid ambiguity without writing too many brackets. We assume that substitution (postfixed) has the highest precedence. Prefixed operators $[x], (x), \nu x, !, \alpha.$ have higher precedence than infixed operators $+$ and $|$.

Lemma 2.3 *Let A be an agent, σ, ρ be two substitutions. If σ and ρ agree on $\mathtt{fn}(A)$, that is $\forall x \in \mathtt{fn}(A).x\sigma$*

2.3 Normalization and Pseudo–Application

We have said earlier that abstractions and concretions have certain standard (normal) forms. Now we define the standard form of an abstraction and that of a concretion. The idea to deal with standard forms first appeared in [Mil91].

Definition 2.7 *An abstraction is in normal form if it is of form $(x)P$. For any abstraction F, its normal form* $\mathtt{norma}(F)$ *is defined inductively on the structure of F as follows:*

1. $\mathtt{norma}((x)P) \equiv (x)P$,
2. *if* $\mathtt{norma}(F) \equiv (y)P$, *then*

$$\mathtt{norma}(F|Q) \equiv (z)(P\iota\{z/y\}|Q) \qquad \mathtt{norma}(Q|F) \equiv (z)(Q|P\iota\{z/y\})$$
$$\mathtt{norma}(\nu x F) \equiv \begin{cases} (y)P & x \notin \mathtt{fn}(F) \\ (x)\nu y P\iota\{x/y, y/x\} & x \in \mathtt{fn}(F) \end{cases}$$

where $z = \mathtt{fresh}(\mathtt{fn}(F, Q))$.

In clause 2. above, it seems that $(y)\nu x P$ would be a simpler definition for $\mathtt{norma}(\nu x F)$ when $x \in \mathtt{fn}(F)$. Our definition swaps x, y in $(y)\nu x P$, which obviously does not change the semantics of the abstraction. However, our choice here is essential for the equivalences in the following Lemma 2.10. If we use the simpler version, those equivalences will only hold for in the sense of α–equivalence.

Definition 2.8 *A concretion is in normal form if it is of form $[x]P$ or $\nu x[x]P$. For any concretion C, its normal form* $\mathtt{normc}(C)$ *is defined inductively on the structure of C as follows:*

1. $\mathtt{normc}([x]P) \equiv [x]P$,
2. *if* $\mathtt{normc}(C) \equiv [y]P$, *then*

$$\mathtt{normc}(C|Q) \equiv [y](P|Q) \qquad \mathtt{normc}(Q|C) \equiv [y](Q|P)$$
$$\mathtt{normc}(\nu x C) \equiv \begin{cases} \nu x[x]P & x = y \\ [y]\nu x P & x \neq y \end{cases}$$

3. *if* $\mathtt{normc}(C) \equiv \nu y[y]P$, *then*

$$\mathtt{normc}(C|Q) \equiv \nu z[z](P\iota\{z/y\}|Q) \qquad \mathtt{normc}(Q|C) \equiv \nu z[z](Q|P\iota\{z/y\})$$
$$\mathtt{normc}(\nu x C) \equiv \begin{cases} \nu y[y]P & x \notin \mathtt{fn}(C) \\ \nu x[x]\nu y P\iota\{x/y, y/x\} & x \in \mathtt{fn}(C) \end{cases}$$

where $z = \mathtt{fresh}(\mathtt{fn}(C, Q))$.

Again we swap the places of x, y in clause . in order to be able to prove the following Lemma 2.10.

Lemma 2.9 *For any abstraction F, concretion C the following hold*

$$\mathtt{fn}(F) = \mathtt{fn}(\mathtt{norma}(F)) \qquad \mathtt{fn}(C) = \mathtt{fn}(\mathtt{normc}(C))$$

Proof It is easy by induction on the structure of F and C. □

Lemma 2.10 *For any abstraction F, concretion C, substitution σ the following hold*

$$\mathtt{norma}(F\sigma) \equiv (\mathtt{norma}(F))\sigma \qquad \mathtt{normc}(C\sigma) \equiv (\mathtt{normc}(C))\sigma$$

Proof Here we only prove the abstraction part. The proof is similar for the concretion part. It is proved by induction on the structure of F. The basic case is that F is in normal form, so $\mathtt{norma}(F) \equiv F$. In this case $F\sigma$ is also in normal form. Thus $\mathtt{norma}(F\sigma) \equiv F\sigma \equiv (\mathtt{norma}(F))\sigma$. For the inductive step suppose $\mathtt{norma}(F\sigma) \equiv (\mathtt{norma}(F))\sigma$ for any σ we will show that

$$\begin{array}{rcl} \mathtt{norma}((\nu x F)\sigma) & \equiv & (\mathtt{norma}(\nu x F))\sigma \\ \mathtt{norma}((F|Q)\sigma) & \equiv & (\mathtt{norma}(F|Q))\sigma \\ \mathtt{norma}((Q|F)\sigma) & \equiv & (\mathtt{norma}(Q|F))\sigma \end{array}$$

We only show the first equivalence. Let $\mathtt{norma}(F) \equiv (y)P$. By the induction hypothesis, for a given name z

$$\mathtt{norma}(F\sigma\{z/x\}) \equiv (\mathtt{norma}(F))\sigma\{z/x\} \equiv ((y)P)\sigma\{z/x\} \equiv (w)P(\sigma\{z/x\})\{w/y\}$$

where $w = \mathtt{fresh}(\mathtt{fn}(((y)P)\sigma\{z/x\})) = \mathtt{fresh}(\mathtt{fn}(F\sigma\{z/x\}))$. So

$$\begin{array}{rcl} & \mathtt{norma}((\nu x F)\sigma) & \\ \equiv & \mathtt{norma}(\nu z F\sigma\{z/x\}) & \\ \equiv & \begin{cases} (w)P(\sigma\{z/x\})\{w/y\} & z \notin \mathtt{fn}(F\sigma\{z/x\}) \\ (z)\nu w P(\sigma\{z/x\})\{w/y\} \circ \iota\{w/z, z/w\} & z \in \mathtt{fn}(F\sigma\{z/x\}) \end{cases} \end{array}$$

where $z = \mathtt{fresh}(\mathtt{fn}((\nu x F)\sigma))$, $w = \mathtt{fresh}(\mathtt{fn}(F\sigma\{z/x\}))$. It is clear that $z \in \mathtt{fn}(F\sigma\{z/x\})$ implies $x \neq y$, so in this case $\mathtt{norma}((\nu x F)\sigma) \equiv (z)\nu w P\sigma\{w/x, z/y\}$. We now need to discuss the following two cases.

The first case is $x \notin \mathtt{fn}(F)$. In this case

$$(\mathtt{norma}(\nu x F))\sigma \equiv ((y)P)\sigma \equiv (w')P\sigma\{w'/y\}$$

where $w' = \mathtt{fresh}(\mathtt{fn}(((y)P)\sigma)) = \mathtt{fresh}(\mathtt{fn}(F\sigma))$. Because $x \notin \mathtt{fn}(F)$ implies $\mathtt{fn}(F\sigma) = \mathtt{fn}(F\sigma\{z/x\})$, so $w = w'$ in this case. Moreover, in this case either $x \notin \mathtt{fn}(P)$ or $x = y$, each of which guarantees $P(\sigma\{z/x\})\{w/y\} \equiv P\{w/y\}$. So $\mathtt{norma}((\nu x F)\sigma) \equiv (\mathtt{norma}(\nu x F))\sigma$.

The second case is $x \in \mathtt{fn}(F)$. In this case $x \neq y$

$$
\begin{aligned}
& (\mathtt{norma}(\nu x F))\sigma \\
\equiv\ & ((x)\nu y P \iota\{x/y, y/x\})\sigma \\
\equiv\ & (z')(\nu y P \iota\{x/y, y/x\})\sigma\{z'/x\} \\
\equiv\ & (z')\nu w' P \iota\{x/y, y/x\} \circ \sigma\{z'/x, w'/y\} \\
\equiv\ & (z')\nu w' P \sigma\{w'/x, z'/y\}
\end{aligned}
$$

and the other wants to receive a name on the same name prot. The result of communication may "twist" the parallel components because of the definition of pseudo–application. We can avoid this by introducing another pseudo–application operator. But this is unnecessary since in any case | will turn out to be a symmetric operator. Restriction νx disallows any communication on the name x. In the rule $name(\alpha)$ gives a singleton set $\{x\}$ when α is x or $\bar{x}$ and gives $\emptyset$ when α is τ. Note that in the rule, the identity substitution ι seems to be unnecessary. However ι may change bound names. Thus although A and $A\iota$ are α-equivalent they are not necessaryly identical. Use of ι here ensures that Rest will not spoil the following Lemma 2.14.

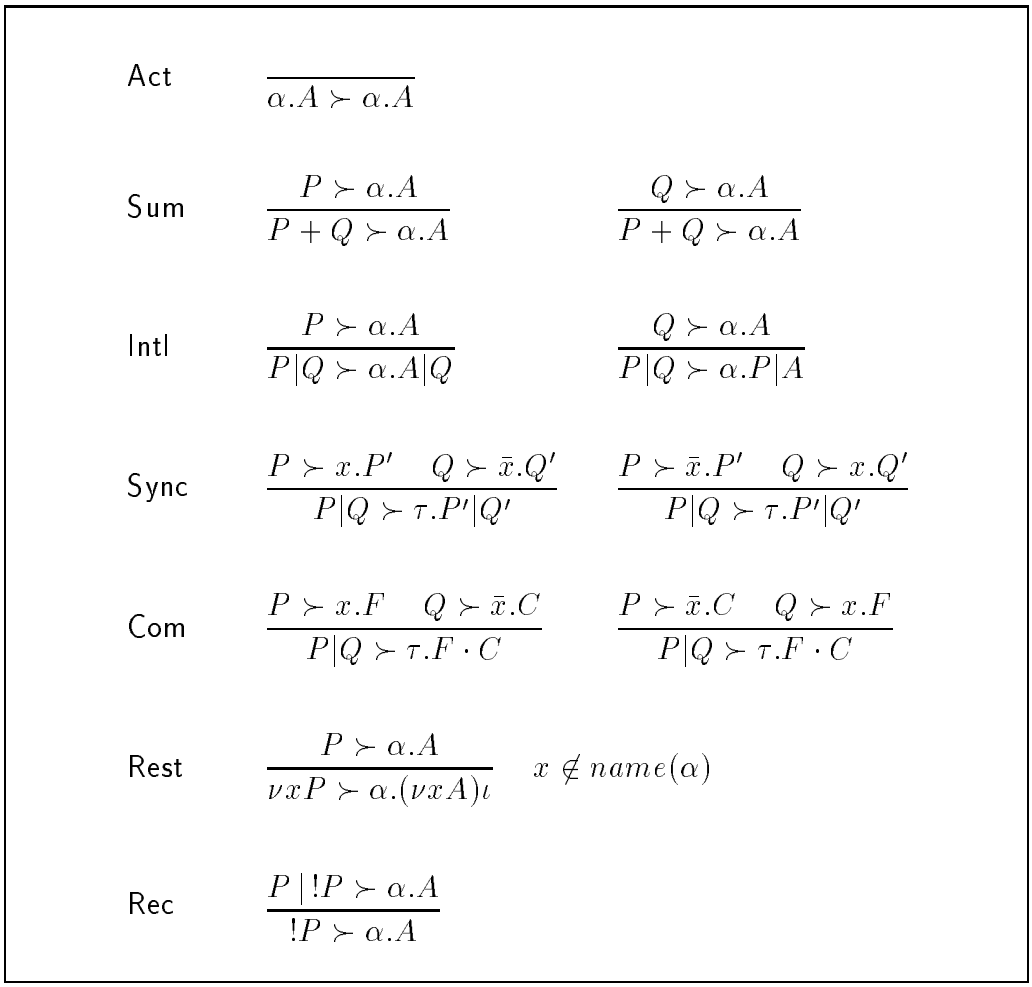

Figure 2: Inference Rules for Commitments

Lemma 2.13 *If $P \succ \alpha.A$ then $name(\alpha) \subseteq \mathtt{fn}(P)$ and $\mathtt{fn}(A) \subseteq \mathtt{fn}(P)$.*

Proof

an example here we check this for Rest. Suppose the property holds for the premiss, that is to say

$$name(\alpha) \subseteq \mathtt{fn}(P) \text{ and } \mathtt{fn}(A) \subseteq \mathtt{fn}(P)$$

we have to show that the property also holds for the conclusion, that is to say

$$name(\alpha) \subseteq \mathtt{fn}(\nu x P) \text{ and } \mathtt{fn}(\nu x A) \subseteq \mathtt{fn}(\nu x P)$$

This immediately follows from the side condition that $x \notin name(\alpha)$. □

Lemma 2.14 *If $P \succ \alpha.A$ then $P\sigma \succ \alpha\sigma.A\sigma$ for any substitution σ.*

Proof Again we only need to show that all the rules in Figure 2

equivalent concretions should have equivalent name parts as well as equivalent process parts. Every thing is quite straightforward until we compare two concretions with bound name parts. In this case what particular bound names are used is not important. What is important is that each of them is always different from any free name. Thus in order for two such concretions to be equivalent, it would be sufficient for the process parts to be equivalent whenever the bound names are replaced by any name different from the free names occuring in the concretions. We can define such an equivalence by the standard notion of bisimulation.

Definition 2.15 *A* strong simulation $\mathcal{S}$ *is a binary relation between agents such that for all* $(A, B) \in \mathcal{S}$*, one of the following must hold:*

1. (A, B) *is a pair of processes* (P, Q) *such that whenever* $P \succ \alpha.A'$ *then* $Q \succ \alpha.B'$ *for some* $(A', B') \in \mathcal{S}$,

2. $\mathtt{norma}(A) \equiv (x)P, \mathtt{norma}(B) \equiv (y)Q$, *and for every name* $z \in \mathcal{N}$

$$(P\iota\{z/x\}, Q\iota\{z/y\}) \in \mathcal{S}$$

3. $\mathtt{normc}(A) \equiv [x]P, \mathtt{normc}(B) \equiv [y]Q, x = y$, *and*

$$(P, Q) \in \mathcal{S}$$

4. $\mathtt{normc}(A) \equiv \nu x[x]P, \mathtt{normc}(B) \equiv \nu y[y]Q$,

for the equivaleefortheprocessthe2.15229.76020Tdet

Definition 2.17 *Two agents A, B are strongly congruent, written $A \sim B$, if $A\sigma \stackrel{\cdot}{\sim} B\sigma$ for all substitutions σ.*

Theorem 2.18 *$\sim$ is a congruence.*

Proof Along the lines in [MPW92a]. □

Theorem 2.19

3 Symbolic D–Bisimulations

This section presents the proof technique for the bisimulation congruence. We introduce a notion of *symbolic D–bisimulation* supported by a proof technique. We then show that the symbolic D–bisimulation coincides with D–bisimulation. Thus the proof technique can be used to show D–bisimulation equivalence of processes, with bisimulation congruence as a special case. The symbolic D–bisimulation introduced here is inspired by *symbolic bisimulation* introduced in [HL92].

The operational semantics introduced in the last section treated free names as constants. This can be seen through the following example. Consider $P|Q$ when $P \succ x.F$ and $Q \succ \bar{y}.C$. If x and y are different names, then the rules of $\succ$ cannot infer any communication between the components (assuming that the components have no other actions). Thus the possibility of communication between these two components when x and y are substituted by the same name is not considered by the relation $\succ$. When names are subject to substitution, it is not sufficient to consider only the $\succ$ relation. To adjust this constant point of view of free names, we introduce *conditional commitment*. We write $P \succ_\sigma \alpha.A$ for conditional commitment which means a commitment under substitution σ. Conditional commitments are defined by the rules in Figure . The rules basically say that conditional commitments are caused by two complementary commitments of parallel components, and that they propagate over the constructions. In the side condition of C-Rest, x is σ clean means $\forall y \in \mathcal{N}.x = y\sigma \Leftrightarrow x = y$. This notation is taken from [Jef92].

Lemma 3.1 *If $P \succ_\sigma \alpha.A$ then $\sigma = \iota\{x/y\}$ for some $x, y \in \mathcal{N}$. Moreover, if $\sigma = \iota\{x/y\}$ where $x \neq y$ then $\alpha = \tau$.*

Proof Easy to check that all the rules in Figure preserve this property. Notice that $\iota = \iota\{x/x\}$ for any $x \in \mathcal{N}$. □

This lemma shows checking for the side condition in rule C-Rest is not difficult at all; the relation $\succ_\sigma$ thus defined is not much more complex than $\succ$. Now we show some desired properties of this relation.

Lemma 3.2 *If $P \succ_\sigma \alpha.A$ then $P\sigma \succ \alpha\sigma.A\sigma$.*

Proof This is to show a property about the relation $\succ_\sigma$ generated by the rules in Figure , we only need to show that all the rules preserve this property. Here we only show this for C-Com and C-Rest.

For the rule

$$P \succ$$

C-Act $$\frac{}{\alpha.A \succ_\iota \alpha.A}$$

C-Sum $$\frac{P \succ_\sigma \alpha.A}{P+Q \succ_\sigma \alpha.A} \qquad \frac{Q \succ_\sigma \alpha.A}{P+Q \succ_\sigma \alpha.A}$$

C-Intl $$\frac{P \succ_\sigma \alpha.A}{P|Q \succ_\sigma \alpha.A|Q} \qquad \frac{Q \succ_\sigma \alpha.A}{P|Q \succ_\sigma \alpha.P|A}$$

C-Sync $$\frac{P \succ_\iota x.P' \quad Q \succ_\iota \bar{y}.Q'}{P|Q \succ_{\iota\{x/y\}} \tau.P'|Q'} \qquad \frac{P \succ_\iota \bar{y}.P' \quad Q \succ_\iota x.Q'}{P|Q \succ_{\iota\{x/y\}} \tau.P'|Q'}$$

C-Com $$\frac{P \succ_\iota x.F \quad Q \succ_\iota \bar{y}.C}{P|Q \succ_{\iota\{x/y\}} \tau.F \cdot C} \qquad \frac{P \succ_\iota \bar{y}.C \quad Q \succ_\iota x.F}{P|Q \succ_{\iota\{x/y\}} \tau.F \cdot C}$$

C-Rest $$\frac{P \succ_\sigma \alpha.A}{\nu x P \succ_\sigma \alpha.(\nu x A)\iota} \quad x \notin name(\alpha),\ x \text{ is } \sigma \text{ clean}$$

C-Rec $$\frac{P \mid !P \succ_\sigma \alpha.A}{!P \succ_\sigma \alpha.A}$$

Figure :

with $A \equiv A'\sigma$, $\alpha = \beta\sigma$, *and for some* ρ', $\sigma = \rho \circ \rho'$.

Proof The direction "if" follows directly from Lemma .2 and Lemma 2.14. The other direction can be proved by induction on the depth of inference and a case analysis of the last rule applied. We only show a key case here.

If $(P|Q)\sigma \succ \alpha.A$, then there are the following possibilities:

1. $P\sigma \succ \alpha.B, A \equiv B|Q\sigma$.

2. $Q\sigma \succ \alpha.B, A \equiv P\sigma|B$.

. $P\sigma \succ x.P', Q\sigma \succ \bar{x}.Q', A \equiv P'|Q'$.

4. $P\sigma \succ \bar{x}.P', Q\sigma \succ x.Q', A \equiv P'|Q'$.

5. $P\sigma \succ x.F, Q\sigma \succ \bar{x}.C, A \equiv F \cdot C$.

6. $P\sigma \succ \bar{x}.C, Q\sigma \succ x.F, A \equiv F \cdot C$.

In the first case, by the induction hypothesis, $P \succ_\rho \beta.B'$ with $\alpha = \beta\sigma, B \equiv B'\sigma$, and for some ρ', $\sigma = \rho \circ \rho'$. So $P|Q \succ_\rho \beta.(B'|Q)$ by C-Intl, and moreover we have $B|Q\sigma \equiv B'\sigma|Q\sigma \equiv (B'|Q)\sigma$.

In the fifth case, by the induction hypothesis,$iretia\psi\psi Tx.QAT\psi\psi T\psi Tte T\psi\psi a\psi T\psi TQT\psi\psi\psi I\psi TQ$

can match $P \succ \alpha.(x|\bar{y})$ *indirectly* by using the fact that $x|\bar{y} \sim_{\{x,y\}} x.\bar{y} + \bar{y}.x$ and $x|\bar{x} \sim x|\bar{x}$. In the following, we will introduce a relation $\succ_{\mathcal{S}}^{D}$ to express this indirect match, where D

so whenever $x, y \in \mathtt{fn}(A, P)$ and $x \neq y$ then $(x, y) \in D$. Because ι respects D, so in this case $P \succ \alpha.B$ for some B with $B \dot{\sim} A$. We can show that for this B, $(A, D, B) \in \mathcal{S}$, that is $A \sim_D B$, by the following series of implications:

$$A \dot{\sim} B \Rightarrow A \sim_{\mathcal{N}} B \Rightarrow A \sim_{\mathcal{N}\lceil \mathtt{fn}(A,B)} B \Rightarrow A \sim_{\mathcal{N}\lceil \mathtt{fn}(A,P)} B \Rightarrow A \sim_D B$$

where the last two implications follow from $\mathcal{N}\lceil \mathtt{fn}(A, B) \subseteq \mathcal{N}\lceil \mathtt{fn}(A, P) \subseteq D$. So in this case $P \succ^D_{\mathcal{S}} \alpha.A$.

Now suppose $x, y \in \mathtt{fn}(A, P), x \neq y$, and $(x, y) \notin D$. It is easy to see that the following holds:

1. for all $\sigma \models D \cup \{x, y\}$, there exists B such that $P\sigma \succ \alpha\sigma.B$ and $B \dot{\sim} A\sigma$, and
2. for all $\sigma \models D$ such that $x\sigma = y\sigma$, there exists B such that $P\sigma \succ \alpha\sigma.B$ and $B \dot{\sim} A\sigma$.

Thus by the induction hypothesis the first implies

$$P \succ_{\mathcal{S}}^{D \cup \{(x,y),(y,x)\}} \alpha.A$$

The second implies for all $\sigma \models D\iota\{y/x\}$, there exists B such that

$$P\iota\{y/x\}\sigma \succ \alpha\sigma.B \text{ and } B \dot{\sim} A\iota\{y/x\}\sigma$$

and by the induction hypothesis this implies

$$P\iota\{y/x\} \succ_{\mathcal{S}}^{D\iota\{y/x\}} \alpha\iota\{y/x\}.A\iota\{y/x\}$$

So by the definition of $\succ_{\mathcal{S}}$, $P \succ^D_{\mathcal{S}} \alpha.A$. □

Now we can give the definition of symbolic D–bisimulation.

Definition 3.7 *A* symbolic simulation, $\mathcal{S}$, *is a set of triples of the form* (A, D, B) *where* A, B *are agents,* D *is a distinction, such that whenever* $(A, D, B) \in \mathcal{S}$, *one of the following must hold:*

1. (A, B) *is a pair of processes* (P, Q) *such that whenever* $P \succ_\sigma \alpha.A'$ *for* $\sigma \models D$ *then* $Q\sigma \succ_{\mathcal{S}}^{D\sigma} \alpha\sigma.A'\sigma$,
2. $\mathtt{norma}(A) \equiv (x)P, \mathtt{norma}(B) \equiv (y)Q$, *and for some* $z \notin \mathtt{fn}(A, B)$
$$(P\iota\{z/x\}, D\backslash z, Q\iota\{z/y\}) \in \mathcal{S}$$
3. $\mathtt{normc}(A) \equiv [x]P, \mathtt{normc}(B) \equiv [y]Q, x = y$, *and*
$$(P, D, Q) \in \mathcal{S}$$
4. $\mathtt{normc}(A) \equiv \nu x[x]P, \mathtt{normc}(B) \equiv \nu y[y]Q$, *and for some* $z \notin \mathtt{fn}(A, B)$
$$(P\iota\{z/x\}, D \cup \{z\} \times \mathtt{fn}(A, B) \cup \mathtt{fn}(A, B) \times \{z\}, Q\iota\{z/y\}) \in \mathcal{S}$$

A set $\mathcal{S}$ of triples is a symbolic bisimulation *if both $\mathcal{S}$ and its inverse $\mathcal{S}^-$ are symbolic simulations. Two agents A, B are said to be* symbolicly D–bisimilar *if there exists a symbolic bisimulation $\mathcal{S}$ such that $(A, D, B) \in \mathcal{S}$.*

In the above definition, we use a set of triples instead of a family of D–indexed sets in order to avoid dealing with set of sets. Theorem .6 suggests that $Q \succ_{\mathcal{S}}^{D} \alpha.A'$ matchs $P \succ \alpha.A'$ in order that $P \sim_D Q$, hence clause 1. In clause 2. z should be viewed as a place holder for any name. A simpler solution would be to choose a z which does not appear free in A, B, and D. However sometimes we may have trouble in choosing such a z: consider $(x)\mathbf{0} \sim_{\mathcal{N}} (y)\mathbf{0}$, we cannot find $z \notin \mathcal{N}$. However, the set of free names of A and B is always finite and any name which appears in D but does not appear free in either A or B is immaterial (Lemma 2.22). With these consideration, clause 2. seems to be workable. In clause 4. z is intended to be a common internal name which takes the place of x in P and y in Q. Thus, not only z should be chosen different from all free names in A and B but also this difference should be remembered in the subsequent reasoning. This is achieved by extending the distinction with the information that z is distinct from all free names in A and B.

Later we will prove that symbolic D-bisimulation coincides with D-bisimulation. The definition of symbolic D-bisimulation is based on the commitments of the processes. Thus it provides us a bisimulation like technique to prove D-bisimulation: in order to prove $P \sim_D Q$, trying to establish a symbolic D-bisimulation $\mathcal{S}$ such that $(P, D, Q) \in \mathcal{S}$. Take the following two processes as given in an earlier example,

$$P \equiv \alpha.(x|\bar{y}) + \alpha.(x.\bar{y} + \bar{y}.x) + \alpha.(x|\bar{x})$$
$$Q \equiv \alpha.(x.\bar{y} + \bar{y}.x) + \alpha.(x|\bar{x})$$

we can prove that $P \sim Q$ by verifying that the following relation $\mathcal{B}$ is a symbolic bisimulation

$$\{(P, \emptyset, Q), (x|\bar{y}, \{x, y\}, x.\bar{y} + \bar{y}.x), (\mathbf{0}|\bar{y}, \{x, y\}, \bar{y}), (x|\mathbf{0}, \{x, y\}, x), (\mathbf{0}|\mathbf{0}, \{x, y\}, \mathbf{0})\} \cup I$$

where $I = \{(A, D, A) \mid \text{agent } A \text{ and distinction } D\}$. In verifying that the above is indeed a symbolic bisimulation, an intersting case is to match $P \succ \alpha.(x|\bar{y})$ with $Q \succ_{\mathcal{B}}^{\emptyset} \alpha.(x|\bar{y})$ which is the consequence of $Q\iota\{y/x\} \succ_{\mathcal{B}}^{\emptyset} \alpha\iota\{y/x\}.(x|\bar{y})\iota\{y/x\}$ and $Q \succ_{\mathcal{B}}^{\{x,y\}} \alpha.(x|\bar{y})$.

In the rest of this section we will prove that the symbolic D–bisimulation indeed characterizes D–bisimulation equivalence.

Theorem 3.8 *If A and B are D–bisimilar, then they are symbolicly D–bisimilar.*

Proof Let

$$\mathcal{S} = \{(A, D, B) \mid A \sim_D B\}$$

Because $\sim_D$ is symmetric, in order to show $\mathcal{S}$ is a symbolic bisimulation it is sufficient to show that $\mathcal{S}$ is a symbolic simulation. Take $(A, D, B) \in \mathcal{S}$, that is to say $A \sim_D B$. Because $\iota \models D$, so $A \dot{\sim} B$. Thus there are the following four cases.

First consider the case that (A, B) is a pair of processes (P, Q). We will show that whenever $P \succ_\sigma \alpha.A'$ for $\sigma \models D$ then $Q\sigma \succ_{\mathcal{S}}^{D\sigma} \alpha\sigma.A'\sigma$. For that, suppose $P \succ_\sigma \alpha.A'$ and $\sigma \models D$, by Theorem .6, we show that for all $\rho \models D\sigma$ there exists B such that $(Q\sigma)\rho \succ (\alpha\sigma)\rho.B$ and $B \dot{\sim} (A'\sigma)\rho$. This is guaranteed by the following:

1. $\rho \models D\sigma$ implies $\sigma \circ \rho \models D$, and thus
2. $P\sigma \circ \rho \dot{\sim} Q\sigma \circ \rho$, and moreover

. $P \succ_\sigma \alpha.A'$ implies $P\sigma \succ \alpha\sigma.A'\sigma$

$P \succ_\rho \beta.A''$, by the definition of $\succ_\rho$, Q must satisfy $Q\rho \succ_{\mathcal{S}}^{D\rho} \beta\rho.A''\rho$. Because $\rho \circ \rho' = \sigma \models D$, it is not difficult to see that $\rho' \models D\rho$. Then by Lemma .5, there exist D', B'' such that $Q\rho \circ \rho' \succ \beta\rho \circ \rho'.B''\rho', \rho' \models D'$ and $(A''\rho, D', B'') \in \mathcal{S}$. So we find $Q\sigma \succ \alpha.B''\rho'$ with $(A''\rho \circ \rho', B''\rho') \in \mathcal{B}$.

If $\mathtt{norma}(A) \equiv (x)P, \mathtt{norma}(B) \equiv (y)Q$, then $\mathtt{norma}(A\sigma) \equiv (u)P\sigma\{u/x\}$ and $\mathtt{norma}(B\sigma) \equiv (v)Q\sigma\{v/y\}$ by Lemma 2.10 and the definition of substitution, where $u = \mathtt{fresh}(\{z\sigma \mid z \in \mathtt{fn}(P) - \{x\}\})$ and $v = \mathtt{fresh}(\{z\sigma \mid z \in \mathtt{fn}(Q) - \{y\}\})$. In this case we have to show that for any name $w \in \mathcal{N}$

$$((P\sigma\{u/x\})\iota\{w/u\}, (Q\sigma\{v/y\})\iota\{w/v\}) \in \mathcal{B}$$

Now because $(A, D, B) \in \mathcal{S}$ and $\mathcal{S}$ is a symbolic bisimulation, so

$$(P\iota\{z/x\}, D\backslash z, Q\iota\{z/y\}) \in \mathcal{S}$$

for some $z \notin \mathtt{fn}(A, B)$. For $\sigma \models D$, it must be the case that $\sigma\{w/z\} \models D\backslash z$. Thus $((P\iota\{z/x\})\sigma\{w/z\}, (Q\iota\{z/y\})\sigma\{w/z\}) \in \mathcal{B}$.

If $\mathtt{normc}(A) \equiv [x]P, \mathtt{normc}(B) \equiv [x]Q$, then by Lemma 2.10 $\mathtt{normc}(A\sigma) \equiv [x\sigma]P\sigma$ and $\mathtt{norma}(B\sigma) \equiv [x\sigma]Q\sigma$. Because $(A, D, B) \in \mathcal{S}$, thus $(P, D, Q) \in \mathcal{S}$. In this case obviously $(P\sigma, Q\sigma) \in \mathcal{B}$.

If $\mathtt{normc}(A) \equiv \nu x[x]P, \mathtt{normc}(B) \equiv \nu y[y]Q$, then $\mathtt{normc}(A\sigma) \equiv \nu u[u]P\sigma\{u/x\}$ and $\mathtt{normc}(B\sigma) \equiv \nu v[v]Q\sigma\{v/y\}$ by Lemma 2.10 and the definition of substitution, where $u = \mathtt{fresh}(\{z\sigma \mid z \in \mathtt{fn}(P) - \{x\}\})$ and $v = \mathtt{fresh}(\{z\sigma \mid z \in \mathtt{fn}(Q) - \{y\}\})$. In this case we have to show that for some $w \notin \mathtt{fn}(A, B)$

$$((P\sigma\{u/x\})\iota\{w/u\}, (Q\sigma\{v/y\})\iota\{w/v\}) \in \mathcal{B}$$

Because $(A, D, B) \in \mathcal{S}$, it follows that

$$(P\iota\{z/x\}, D \cup \{z\} \times \mathtt{fn}(A, B) \cup \mathtt{fn}(A, B) \times \{z\}, Q\iota\{z/y\}) \in \mathcal{S}$$

by the definition of symbolic D–bisimulation. Now let $w \notin \{x\sigma \mid x \in \mathtt{fn}(A, B)\}$, then it is clear that $\sigma\{w/z\} \models D \cup \{z\} \times \mathtt{fn}(A, B) \cup \mathtt{fn}(A, B) \times \{z\}$. Thus $((P\iota\{z/x\})\sigma\{w/z\}, (Q\iota\{z/y\})\sigma\{w/z\}) \in \mathcal{S}$. □

4 Conclusion and Related Work

This paper presents a new characterization of the bisimulation congruence and D–bisimulation equivalences of the π–calculus. The new characterization supports a bisimulation like proof technique which avoids explicit case analysis, thus providing a new way of proving bisimulation congruence.

The proof technique resembles the symbolic bisimulation for value passing processes introduced by Hennessy and Lin [HL92]. In their work, symbolic bisimulation of value passing processes is defined in terms of *symbolic transition* of the form $T \xrightarrow{b,a} T'$, where T, T' are process terms (may have free variables), a is some action, b is a boolean expression. This can be read as "under condition b, T may

perform a and ivolve into T'".

[HL92] M. Hennessy and H. Lin. Symbolic bisimulation. Technical Report Technical Report 1/92, School of Congnitive and Computing Sciences, University of Sussex, 1992.

[Jef92] A. Jeffrey. Notes on a trace semantics for the π–calculus. 1992.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice–Hall, 1989.

[Mil91] R. Milner. The polyadic π–calculus: a tutorial. *The Proceedings of the International Summer School on Logic and Algebra of Specification*, 1991.

[MPW92a] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (parts i and ii). *Information and Computation*, 100, 1992.

[MPW92b] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 1992. to appear.

[Par81] D. Park. Concurrency and automata on infinite sequences. *Lecture Notes In Computer Science, Springer Verlag*, 104, 1981. Proceedings of 5th GI Conference.

[PS9] J. Parrow and Davide Sangiorgi. Algebraic theories for value–passing calculi. Technical report, Department of Computer Science, University of Edinburgh, 199 . Forthcoming.

[San9] D. Sangiorgi. A theory of bisimulation for π–calculus. Technical report, University of Edinburgh, 199 . Forthcoming.

[Sto88] Allen Stoughton. Substitution revisited. *Theoretical Computer Science*, 59: 17– 25, 1988.