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 of bisimulation in CCS with a pleasant proof technique. However, ground bisimulation equivalence is not a congruence relation for π–calculus constructions, because now names are subject to substitution and ground bisimulation equivalence is not preserved under substitution of names. To obtain a congruence relation, bisimulation congruence is defined such that two processes are related just in case they are ground bisimilar under all substitutions. Although this immediately gives us a congruence relation, this definition does not suggest any direct proof technique to establish congruence between processes other than tedious exhaustive case analysis. Thus, in extending CCS to the π–calculus, the nice feature of proof technique is somewhat lost for bisimulation congruence — the more important relation between π–calculus processes. In fact there is a whole series of distinction bisimulation equivalences such that two processes are D–bisimilar just in case they are ground bisimilar under all substitutions respecting D, where D is the so called *distinction* which tells that some pairs of names should not be substituted into the same name. Here the general definition of D–bisimilarity again is quantified over certain substitutions, and therefore it does not provide any proof technique to check D–bisimilarity between processes. Notice that bisimulation congruence is a particular instance of D–bisimulation equivalence where D poses no constraint upon substitutions, and ground bisimulation equivalence is a particular instance of D–bisimulation equivalence where D distinguishes any pair of names.

The purpose of this paper is to introduce a proof technique for bisimulation congruence of π–calculus. We characterize bisimulation congruence by an alternative definition based directly on the operational behaviour of processes. The new definition is in a style of bisimulation without explicit case analysis, thus providing a bisimulation like proof technique for congruence. Generally, we will find that we cannot do this for the bisimulation congruence alone, we have to do it for D–bisimulation equivalences all together. Thus the proof technique can be used to establish D–bisimulation equivalences for any distinction D.

Here we roughly illustrate the idea behind our approach. Let us write $\sim$ for bisimulation congruence, $\sim_D$ for D–bisimulation equivalence, and $\dot{\sim}$ for ground bisimulation equivalence. A basic fact we can exploit in proving congruence of two processes is the following:

$$P \sim Q \text{ if and only if } P \sim_{\{x,y\}} Q \text{ and } P\{y/x\} \sim Q\{y/x\}$$

where x, y are two different given names, $\{x, y\}$ is the distinction which tells that x and y must not be substituted into the same name, and $P\{y/x\}$ means the term obtained by substituting y for free occurrences of x in P. This fact holds for the following reason. Because $P \sim Q$ means $P\sigma \dot{\sim} Q\sigma$ for any substitution σ, $P \sim_{\{x,y\}} Q$ means $P\sigma \dot{\sim} Q\sigma$ for those substitution σ that $x\sigma \neq y\sigma$, and $P\{y/x\} \sim Q\{y/x\}$ means $P\{y/x\}\sigma \dot{\sim} Q\{y/x\}\sigma$ for any substitution σ or equivalently $P\sigma \dot{\sim} Q\sigma$ for any substitution σ that $x\sigma = y\sigma$. Thus for two given names x, y, $P \sim Q$ just in case $P \sim_{\{x,y\}} Q$ and $P\{y/x\} \sim Q\{y/x\}$. Using this fact, if we choose properly x

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 should be shown to behave properly in the presence of α–equivalence. Previous works on the theory of π–calculus, like [MPW92a, MPW92b, San93], all employ the notion of α–equivalence in the semantic definition, thus the following development of the semantic theory suffer from the complication caused by α–equivalence. In this paper, by using the notion of *simultanous substitution* introduced in [Sto88], we show that the semantics of π–calculus can be defined on the much simpler notion of syntactic identity instead of α–equivalence. This gives a considerable advantage in the development of the theory.

The next section presents the basic algegbraic theory of the π-calculus. The section follows that presents the alternative characterization of bisimulation congruence and D-bisimulation equivalence. Section 4 concludes with a short discussion about some related work.

2 The π–Calculus

This section is a presentation of the basic algebraic theory of the π–calculus developed in [MPW92a, Mil91]. The presentation here is tailored to fit the new characterization of bisimulation congruence to be introduced in the next section. The reader is referred to the original papers for a general account of the motivation and background of the π–calculus.

2.1 Agents

The π–calculus presented here is a simplified version of that in [Mil91]. The primitive entity in the π–calculus is a *name* which has no structure. We assume that we have an infinite set of names $\mathcal{N}$, and we use $x, y, \ldots$ to range over it. The main entity in the π–calculus is an *agent*, which can be a *process*, an *abstraction*, or a *concretion*. We use $P, Q, \ldots$ to range over processes, $F, G, \ldots$ over abstractions, $C, D, \ldots$ over concretions, and $A, B, \ldots$ over agents. These are built from names by the syntax in Figure 1.

$$\boxed{\begin{array}{lcl} P & ::= & \mathbf{0} \mid \tau.P \mid x.P \mid \bar{x}.P \mid x.F \mid \bar{x}.C \mid P+P' \mid P|P' \mid !P \mid \nu x P \\ F & ::= & (x)P \mid F|P \mid P|F \mid \nu x F \\ C & ::= & [x]P \mid C|P \mid P|C \mid \nu x C \\ A & ::= & P \mid F \mid C \end{array}}$$

Figure 1: Agents of the π–calculus

We assume that the reader has some familiarity with CCS [Mil89]; here we give an informal description of the above constructions and the intended meaning by comparing the process constructions here with those in CCS. Roughly speaking, π–calculus is a generalization of value passing CCS in that names may be passed in communication between processes. $\mathbf{0}, \tau.P, x.P, \bar{x}.P, P+Q, P|Q$ are the constructions inherited from CCS. $\nu x P$ is $P \backslash x$ in CCS notation, which restricts the use of the name x to P. So, νx binds free occurrences of x in P. A concretion C consists of two parts: a process and a name. The name part of $[x]P$ is a free name x while 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 $\mathtt{fresh}$ 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 $\mathtt{fresh}(N)$ 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:

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

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 $|$. And $|$ has higher precedence than $+$. Thus $\nu z[z]P\sigma\{z/x\}$ means $\nu z[z](P\sigma\{z/x\})$. Throughout the paper, $\equiv$ is used for syntactic identity. One of the main advantages of simultaneous substitution is to enable us to work with syntactic identity, avoiding the troublesome α–equivalence.

The following are some properties enjoyed by this approach of substitution.

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

Proof Induction on the structure of A. $\Box$

Corollary 2.4 *Let A be an agent, σ, ρ be substitutions, w, x, z be names. If $z \notin \{y\sigma \mid y \in \mathtt{fn}(A) - \{x\}\}$ then $A(\sigma \circ \rho)\{w/x\} \equiv A\sigma\{z/x\} \circ \rho\{w/z\}$.*

Proof Because $z \notin \{y\sigma \mid y \in \mathtt{fn}(A) - \{x\}\}$ implies that $\sigma\{z/x\} \circ \rho\{w/z\}$ and $(\sigma \circ \rho)\{w/x\}$ agree on $\mathtt{fn}(A)$. $\Box$

This corollary will be repeatedly used in the rest of the paper, sometimes with $\sigma = \iota$ or $\rho = \iota$.

Lemma 2.5 $\mathtt{fn}(A\sigma) = \{x\sigma \mid x \in \mathtt{fn}(A)\}$.

Proof Induction on the structure of A. $\Box$

By this lemma, it is easy to see that $\mathtt{fn}((A\sigma)\rho) = \mathtt{fn}(A\sigma \circ \rho)$, because the left hand side $\{y\rho \mid y \in \mathtt{fn}(A\sigma)\} = \{y\rho \mid y \in \{x\sigma \mid x \in \mathtt{fn}(A)\}\}$ equals to the right hand side which is $\{x\sigma \circ \rho \mid x \in \mathtt{fn}(A)\} = \mathtt{fn}(A\sigma \circ \rho)$. Using this fact, we can prove the following stronger result.

Lemma 2.6 *Let A be an agent, σ, ρ be two substitutions. Then $(A\sigma)\rho \equiv A\sigma \circ \rho$.*

Proof Induction on the structure of A. As an example we show

$$((\nu x P)\sigma)\rho \equiv (\nu x P)\sigma \circ \rho$$

By definition 2.2,

$$(\nu x P)\sigma \circ \rho \equiv \nu z P(\sigma \circ \rho)\{z/x\}$$

where $z = \mathtt{fresh}(\mathtt{fn}((\nu x P)\sigma \circ \rho))$ (using lemma 2.5). By definition 2.2 and the induction hypothesis,

$$((\nu x P)\sigma)\rho \equiv (\nu v P\sigma\{v/x\})\rho \equiv \nu w(P\sigma\{v/x\})\rho\{w/v\} \equiv \nu w P\sigma\{v/x\} \circ \rho\{w/v\}$$

where $w = \mathtt{fresh}(\mathtt{fn}((\nu v P\sigma\{v/x\})\rho)) = \mathtt{fresh}(\mathtt{fn}(((\nu x P)\sigma)\rho))$ and $v = \mathtt{fresh}(\{y\sigma \mid y \in \mathtt{fn}(P) - \{x\}\})$. As we have argued that by Lemma 2.5 $w = z$. Moreover, because $v \notin \{y\sigma \mid y \in \mathtt{fn}(P) - \{x\}\}$, by Corollary 2.4

$$\nu z P\sigma\{v/x\} \circ \rho\{z/v\} \equiv \nu z P(\sigma \circ \rho)\{z/x\}$$

hence $((\nu x P)\sigma)\rho \equiv (\nu x P)\sigma \circ \rho$. $\Box$

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 $\texttt{norma}(F)$ is defined inductively on the structure of F as follows:*

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

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

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

In clause 2. above, it seems that $(y)\nu xP$ would be a simpler definition for $\texttt{norma}(\nu xF)$ when $x \in \texttt{fn}(F)$. Our definition swaps x, y in $(y)\nu xP$, 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 $\texttt{normc}(C)$ is defined inductively on the structure of C as follows:*

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

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

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

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

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

Again we swap the places of x, y in clause 3. 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. $\square$

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

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

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

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

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 $x \in \texttt{fn}(F)$. In this case $x \neq y$

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

where $z' = \texttt{fresh}(\texttt{fn}(((x)\nu y P\iota\{x/y, y/x\})\sigma))$ and $w' = \texttt{fresh}(\texttt{fn}((\nu y P\iota\{x/y, y/x\})\sigma\{z'/x\}))$. Now we only need to show $w = w'$ and $z = z'$ in order to establish $\texttt{norma}((\nu x F)\sigma) \equiv (\texttt{norma}(\nu x F))\sigma$ in this case. It is not difficult to work out that $\texttt{fn}((x)\nu y P\iota\{x/y, y/x\}) = \texttt{fn}(\nu x F)$ and $\texttt{fn}(\nu y P\iota\{x/y, y/x\}) = \texttt{fn}(F)$, which guarntee $w' = w$ and $z' = z$. $\square$

In normal forms, it is clear that an abstraction is ready to receive a name, and a concretion is ready to give a name. In [Mil91], a *pseudo–application* operator $\cdot$ is introduced to describe the result of the action of passing the name part of a concretion to an abstraction.

Definition 2.11 *Pseudo–application $\cdot$ is a binary operator between abstractions and concretions defined as follows,*

1. *If $\texttt{norma}(F) \equiv (x)P$ and $\texttt{normc}(C) \equiv [y]Q$ then*

   $F \cdot C$ is defined to be $P\iota\{y/x\}|Q$

2. *If $\texttt{norma}(F) \equiv (x)P$ and $\texttt{normc}(C) \equiv \nu y[y]Q$ then*

   $F \cdot C$ is defined to be $\nu z(P\iota\{z/x\}|Q\iota\{z/y\})$

   where $z = \texttt{fresh}(\texttt{fn}(F, C))$.

In the above, when the name part of the concretion is a free name, this name is passed by simply substituting all free occurrences of x in P. To pass a bound name means to let the internal name be known by the process which receives the name, but it should not be known by other processes. So by passing a bound name, the scope of the restriction is extended to include the process receiving it. This is called *scope extrusion.*

Lemma 2.12 *The following equivalence holds for any substitution σ*

$$(F \cdot C)\sigma \equiv F\sigma \cdot C\sigma$$

Proof If $\texttt{norma}(F) \equiv (x)P$ and $\texttt{normc}(C) \equiv [y]Q$, then the conclusion directly follows from Lemma 2.10. Here we show that it is true when $\texttt{norma}(F) \equiv (x)P$ and $\texttt{normc}(C) \equiv \nu y[y]Q$. In this case we have

$$\begin{aligned} & (F \cdot C)\sigma \\ \equiv\ & (\nu z(P\iota\{z/x\}|Q\iota\{z/y\}))\sigma \\ \equiv\ & \nu w(P\iota\{z/x\}|Q\iota\{z/y\})\sigma\{w/z\} \\ \equiv\ & \nu w(P\iota\{z/x\} \circ \sigma\{w/z\}|Q\iota\{z/y\} \circ \sigma\{w/z\}) \end{aligned}$$

where $w = \texttt{fresh}(\texttt{fn}((\nu z(P\iota\{z/x\}|Q\iota\{z/y\}))\sigma))$, and $z = \texttt{fresh}(\texttt{fn}(F, C))$. By Lemma 2.10,

$$\begin{aligned} \texttt{norma}(F\sigma) &\equiv (\texttt{norma}(F))\sigma \equiv ((x)P)\sigma \equiv (u)P\sigma\{u/x\} \\ \texttt{normc}(C\sigma) &\equiv (\texttt{normc}(C))\sigma \equiv (\nu y[y]Q)\sigma \equiv \nu v[v]Q\sigma\{v/y\} \end{aligned}$$

where $u = \texttt{fresh}(\texttt{fn}(F\sigma)), v = \texttt{fresh}(\texttt{fn}(C\sigma))$. Thus

$$F\sigma \cdot C\sigma \equiv \nu w'(P\sigma\{u/x\} \circ \iota\{w'/u\}|Q\sigma\{v/y\} \circ \iota\{w'/v\})$$

where $w' = \texttt{fresh}(\texttt{fn}(F\sigma, C\sigma))$. By Lemma 2.9,

$$\texttt{fn}(F\sigma, C\sigma) = \texttt{fn}(\texttt{norma}(F\sigma), \texttt{normc}(C\sigma)) = \texttt{fn}(((x)P)\sigma, (\nu y[y]Q)\sigma)$$

Thus $w = w'$ because $\texttt{fn}((\nu z(P\iota\{z/x\}|Q\iota\{z/y\}))\sigma) = \texttt{fn}(((x)P)\sigma, (\nu y[y]Q)\sigma)$. For the rest we only need to show that $\iota\{z/x\} \circ \sigma\{w/z\}$ and $\sigma\{u/x\} \circ \iota\{w/u\}$ agree on free names of P, and $\iota\{z/y\} \circ \sigma\{w/z\}$ and $\sigma\{v/y\} \circ \iota\{w/v\}$ agree on free names of Q. It is obvious that $\iota\{z/x\} \circ \sigma\{w/z\}$ and $\sigma\{u/x\} \circ \iota\{w/u\}$ agree on x, because both give w. Suppose t is a free name of P other than x, then it must be the case that $t \in \texttt{fn}((x)P) = \texttt{fn}(F)$. Thus $t \neq z$ and $t\sigma \neq u$. So in this case both $t\iota\{z/x\} \circ \sigma\{w/z\}$ and $t\sigma\{u/x\} \circ \iota\{w/u\}$ gave $t\sigma$. In the same way we can show that $\iota\{z/y\} \circ \sigma\{w/z\}$ and $\sigma\{v/y\} \circ \iota\{w/v\}$ agree on free names of Q. □

2.4 Commitments

To formally describe the semantics of agents, we adopt the notion of *commitment* introduced in [Mil91]. A commitment $\alpha.A$ consists of an *action* α and a *continuation* A which is an agent. There are three kinds of α's: internal action τ, input action x via a name x, and output action $\bar{x}$ via a name x. Now the behaviour of a process can be described by its commitments. The way it is formalized is to define the relation

$$P \succ \alpha.A$$

between processes and commitments. It is clear that the set of commitments of $P + Q$ should be the union of that of P and Q. The relation $\succ$ is defined by the rules shown in Figure 2. Most of the rules explain themselves. Communication happens between two parallel components only when one wants to output a name 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.

$$
\begin{array}{lll}
\textsf{Act} & \dfrac{}{\alpha.A \succ \alpha.A} & \\[2ex]
\textsf{Sum} & \dfrac{P \succ \alpha.A}{P + Q \succ \alpha.A} & \dfrac{Q \succ \alpha.A}{P + Q \succ \alpha.A} \\[2ex]
\textsf{Intl} & \dfrac{P \succ \alpha.A}{P|Q \succ \alpha.A|Q} & \dfrac{Q \succ \alpha.A}{P|Q \succ \alpha.P|A} \\[2ex]
\textsf{Sync} & \dfrac{P \succ x.P' \quad Q \succ \bar{x}.Q'}{P|Q \succ \tau.P'|Q'} & \dfrac{P \succ \bar{x}.P' \quad Q \succ x.Q'}{P|Q \succ \tau.P'|Q'} \\[2ex]
\textsf{Com} & \dfrac{P \succ x.F \quad Q \succ \bar{x}.C}{P|Q \succ \tau.F \cdot C} & \dfrac{P \succ \bar{x}.C \quad Q \succ x.F}{P|Q \succ \tau.F \cdot C} \\[2ex]
\textsf{Rest} & \dfrac{P \succ \alpha.A}{\nu x P \succ \alpha.(\nu x A)\iota} & x \notin name(\alpha) \\[2ex]
\textsf{Rec} & \dfrac{P \,|\, !P \succ \alpha.A}{!P \succ \alpha.A} &
\end{array}
$$

Figure 2: Inference Rules for Commitments

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

Proof This is to show a property of the relation $\succ$ which is generated by the rules in Figure 2, thus we only need to show that all the rules preserve this property. As 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 preserve this property. Here we only show this for Com and Rest.

For Com, suppose for any substitution σ the following implication hold

$$P \succ x.F \Rightarrow P\sigma \succ x\sigma.F\sigma$$
$$Q \succ \bar{x}.C \Rightarrow Q\sigma \succ \bar{x}\sigma.C\sigma$$

we will show that for any substitution σ,

$$P|Q \succ \tau.F \cdot C \Rightarrow (P|Q)\sigma \succ \tau\sigma.(F \cdot C)\sigma$$

Notice that $\bar{x}\sigma = \overline{x\sigma}$, $\tau\sigma = \tau$, and by Lemma 2.12 $(F \cdot C)\sigma \equiv F\sigma \cdot C\sigma$. Thus $(P|Q)\sigma \succ \tau\sigma.(F \cdot C)\sigma$ follows immediately from rule Com.

For Rest, suppose $P \succ \alpha.A$ implies $P\sigma \succ \alpha\sigma.A\sigma$ for any σ, we will show that $\nu x P \succ \alpha.(\nu x A)\iota$ implies $(\nu x P)\sigma \succ \alpha\sigma.(\nu x A)\sigma$ for any substitution σ. Take any σ, and let $z = \mathtt{fresh}(\{y\sigma \mid y \in \mathtt{fn}(\nu x P)\})$. By the induction hypothesis we have

$$P\sigma\{z/x\} \succ \alpha\sigma\{z/x\}.A\sigma\{z/x\}$$

Since Rest is applicable, it follows that $x \notin name(\alpha)$ and thus $\alpha\sigma\{z/x\} = \alpha\sigma$. Because $z = \mathtt{fresh}(\mathtt{fn}((\nu x P)\sigma))$, and by the previous lemma $name(\alpha) \subseteq \mathtt{fn}(\nu x P)$. Thus $z \notin name(\alpha\sigma)$, and we can apply Rest to obtain

$$\nu z P\sigma\{z/x\} \succ \alpha\sigma.(\nu z A\sigma\{z/x\})\iota$$

By our choice of z, it is clear that $(\nu x P)\sigma = \nu z P\sigma\{z/x\}$. Now we only need to show $(\nu x A)\sigma = (\nu z A\sigma\{z/x\})\iota$ in order to establish the case. The presence of ι guarantees this. □

2.5 Ground Bisimulation and Congruence

We now go on to define an equivalence relation for agents. Let us first look at some expected properties of this equivalence relation. For two equivalent processes, every commitment of a process should be matched by an *equivalent* commitment of the other process. Thanks to the normalization of abstraction and concretion, two equivalent abstractions should gave equivalent processes for any name. Two 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. *$\texttt{norma}(A) \equiv (x)P, \texttt{norma}(B) \equiv (y)Q$, and for every name $z \in \mathcal{N}$*
$$(P\iota\{z/x\}, Q\iota\{z/y\}) \in \mathcal{S}$$
3. *$\texttt{normc}(A) \equiv [x]P, \texttt{normc}(B) \equiv [y]Q, x = y$, and*
$$(P, Q) \in \mathcal{S}$$
4. *$\texttt{normc}(A) \equiv \nu x[x]P, \texttt{normc}(B) \equiv \nu y[y]Q$, and for some $z \notin \texttt{fn}(A, B)$*
$$(P\iota\{z/x\}, Q\iota\{z/y\}) \in \mathcal{S}$$

A binary relation $\mathcal{S}$ is a (strong) bisimulation *if both $\mathcal{S}$ and its inverse are strong simulations. We say that A is* strong ground bisimilar *to B if there exists a bisimulation $\mathcal{S}$ such that $(A, B) \in \mathcal{S}$. In this case we write $A \dot{\sim} B$.*

It is routine to show that $\dot{\sim}$ is an equivalence relation. However it is not preserved by substitution. As a simple counter–example

$$\bar{x}.\mathbf{0}|y.\mathbf{0} \dot{\sim} \bar{x}.y.\mathbf{0} + y.\bar{x}.\mathbf{0}$$

but obviously

$$(\bar{x}.\mathbf{0}|y.\mathbf{0})\iota\{y/x\} \not\dot{\sim} (\bar{x}.y.\mathbf{0} + y.\bar{x}.\mathbf{0})\iota\{y/x\}$$

because the process on the left has the commitment $\tau.\mathbf{0}$ while the process on the left has not. So

$$(x)(\bar{x}.\mathbf{0}|y.\mathbf{0}) \not\dot{\sim} (x)(\bar{x}.y.\mathbf{0} + y.\bar{x}.\mathbf{0})$$

Thus $\dot{\sim}$ is not a congruence.

Lemma 2.16 *If $A \dot{\sim} B$ and $w \notin \texttt{fn}(A, B)$, then $A\iota\{w/z\} \dot{\sim} B\iota\{w/z\}$.*

Proof Similar to the proof in [MPW92a]. □

The following definition gives us a congruence.

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

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

Proof Along the lines in [MPW92a]. $\square$

Theorem 2.19 *For any abstraction F and any concretion C*

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

Proof First, for any abstraction F and any concretion C

$$F \dot\sim \mathtt{norma}(F) \qquad C \dot\sim \mathtt{normc}(C)$$

This is because $I \cup \{(F, \mathtt{norma}(F))\}$ and $I \cup \{(C, \mathtt{normc}(C))\}$ are two bisimulations, where $I = \{(A, A) \mid A \text{ is an agent}\}$. Now it follows that for any substitution σ

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

By Lemma 2.10, $\mathtt{norma}(F\sigma) \equiv (\mathtt{norma}(F))\sigma$ and $\mathtt{normc}(C\sigma) \equiv (\mathtt{normc}(C))\sigma$. Thus for any substitution σ

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

That is

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

$\square$

2.6 Distinctions

Now we will see that a spectrum of equivalences can be defined parameterized by *distinctions*.

Definition 2.20 *A distinction is a symmetric irreflexive relation between names. We shall let D range over distinctions. A substitution σ respects a distinction D, written $\sigma \models D$, if for all $(x, y) \in D, x\sigma \neq y\sigma$.*

By this definition, ι respects any distinction. Here we use some abbreviations introduced in [MPW92a]. We will sometimes write a set of names A for the distinction $\{(x, y) \mid x, y \in A, x \neq y\}$ which keeps all members of A distinct from each other. Also for a distinction D and a name x, we define

$$D\backslash x = D - (\{x\} \times \mathcal{N} \cup \mathcal{N} \times \{x\})$$

This removes any constraint in D upon the substitution for x. Also, for any set $A \subseteq \mathcal{N}$ of names, we define $D\lceil A = D \cap (A \times A)$, and for any substitution σ which respects D, we define $D\sigma = \{(x\sigma, y\sigma) \mid (x, y) \in D\}$.

Definition 2.21 *Two agents A and B are D–*bisimilar*, written $A \sim_D B$, if $A\sigma \dot{\sim} B\sigma$ for all substitutions σ respecting D.*

It is easy to see that $\sim_D$ is an equivalence relation for any D, and that $\sim_D \subseteq \sim_{D'}$ just in case $D' \subseteq D$. In this sense, ground bisimulation equivalence and bisimulation congruence are two extreme cases of D–bisimulations

$$\dot{\sim} = \sim_{\mathcal{N}} \text{ and } \sim = \sim_{\emptyset}$$

To finish this section, we prove some properties of D–bisimulation which are important later in finding an alternative characterization of it.

Lemma 2.22 *If $P \sim_D Q$ then $P \sim_{D\lceil \mathtt{fn}(P,Q)} Q$.*

Proof Along the lines in [MPW92a]. $\square$

Proposition 2.23 *If $[x]P \sim_D [x]Q$ then $P \sim_D Q$.*

Proof We have to show that $P\sigma \dot{\sim} Q\sigma$ whenever σ respects D. Because $[x]P \sim_D [x]Q$, so $([x]P)\sigma \dot{\sim} ([x]Q)\sigma$. That is $[x\sigma]P\sigma \dot{\sim} [x\sigma]Q\sigma$. By definition 2.15 this implies $P\sigma \dot{\sim} Q\sigma$. $\square$

Proposition 2.24 *If $(x)P \sim_D (y)Q$ and $z \notin \mathtt{fn}((x)P, (y)Q)$, then*

$$P\iota\{z/x\} \sim_{D\backslash z} Q\iota\{z/y\}$$

Proof We will show that $(P\iota\{z/x\})\sigma \dot{\sim} (Q\iota\{z/y\})\sigma$ if σ respects $D\backslash z$.

As a first step we will show $((x)P)\sigma \dot{\sim} ((y)Q)\sigma$. Because $z \notin \mathtt{fn}((x)P,(y)Q)$, so $D\backslash z \supseteq D\lceil \mathtt{fn}((x)P,(y)Q)$ thus σ respects $D\lceil \mathtt{fn}((x)P,(y)Q)$. On the other hand $(x)P \sim_D (y)Q$ implies $(x)P \sim_{D\lceil \mathtt{fn}((x)P,(y)Q)} (y)Q$, so $((x)P)\sigma \dot{\sim} ((y)Q)\sigma$.

Now consider $((x)P)\sigma \dot{\sim} ((y)Q)\sigma$. By definition 2.2

$$(u)P\sigma\{u/x\} \dot{\sim} (v)Q\sigma\{v/y\}$$

where $u = \mathtt{fresh}(\{w\sigma \,|\, w \in \mathtt{fn}(P) - \{x\}\}), v = \mathtt{fresh}(\{w\sigma \,|\, w \in \mathtt{fn}(Q) - \{y\}\})$. By definition 2.15 this implies $(P\sigma\{u/x\})\iota\{z\sigma/u\} \dot{\sim} (Q\sigma\{v/y\})\iota\{z\sigma/v\}$. Thus follows from Lemma 2.6 that

$$P\sigma\{u/x\} \circ \iota\{z\sigma/u\} \dot{\sim} Q\sigma\{v/y\} \circ \iota\{z\sigma/v\}$$

Now because $u \notin \{w\sigma \mid w \in \mathtt{fn}(P) - \{x\}\}$, by Corollary 2.4

$$P\sigma\{z\sigma/x\} \equiv P\sigma\{u/x\} \circ \iota\{z\sigma/u\}$$

Likewise $Q\sigma\{z\sigma/y\} \equiv Q\sigma\{v/y\} \circ \iota\{z\sigma/v\}$. So $P\sigma\{z\sigma/x\} \dot{\sim} Q\sigma\{z\sigma/y\}$. Now because $\sigma\{z\sigma/x\} = \iota\{z/x\} \circ \sigma$ and $\sigma\{z\sigma/y\} = \iota\{z/y\} \circ \sigma$, we have

$$(P\iota\{z/x\})\sigma \dot{\sim} (Q\iota\{z/y\})\sigma$$

$\square$

Proposition 2.25 *If $\nu x[x]P \sim_D \nu y[y]Q$ and $z \notin \mathtt{fn}(\nu x[x]P, \nu y[y]Q)$, then*

$$P\iota\{z/x\} \sim_{D \cup D'} Q\iota\{z/y\}$$

where $D' = \{z\} \times \mathtt{fn}(\nu x[x]P, \nu y[y]Q) \cup \mathtt{fn}(\nu x[x]P, \nu y[y]Q) \times \{z\}$.

Proof Let σ respect $D \cup D'$, we will show that $(P\iota\{z/x\})\sigma \dot{\sim} (Q\iota\{z/y\})\sigma$.

As the first step we show that there exists $w \notin \mathtt{fn}((\nu x[x]P)\sigma, (\nu y[y]Q)\sigma)$ such that

$$P\sigma\{w/x\} \dot{\sim} Q\sigma\{w/y\}$$

We start from $(\nu x[x]P)\sigma \dot{\sim} (\nu y[y]Q)\sigma$ since it is obvious that σ respects D. By definition 2.2

$$\nu u[u]P\sigma\{u/x\} \dot{\sim} \nu v[v]Q\sigma\{v/y\}$$

where $u = \mathtt{fresh}(\{w\sigma \mid w \in \mathtt{fn}(P) - \{x\}\}), v = \mathtt{fresh}(\{w\sigma \mid w \in \mathtt{fn}(Q) - \{y\}\})$. By definition 2.15 this implies

$$(P\sigma\{u/x\})\iota\{w/u\} \dot{\sim} (Q\sigma\{v/y\})\iota\{w/v\}$$

for some $w \notin \mathtt{fn}((\nu x[x]P)\sigma, (\nu y[y]Q)\sigma)$. Because $u \notin \{w\sigma \mid w \in \mathtt{fn}(P) - \{x\}\}$, by Corollary 2.4 $P\sigma\{w/x\} \equiv (P\sigma\{u/x\})\iota\{w/u\}$, and $Q\sigma\{w/y\} \equiv (Q\sigma\{v/y\})\iota\{w/v\}$. So $P\sigma\{w/x\} \dot{\sim} Q\sigma\{w/y\}$.

As the second step we show that

$$P\sigma\{z\sigma/x\} \dot{\sim} Q\sigma\{z\sigma/y\}$$

If $w = z\sigma$ it directly follows from the last step. So in the following we assume $w \neq z\sigma$. Because σ respects D', and with lemma 2.6 it is not difficult to see that this implies $z\sigma \notin \mathtt{fn}((\nu x[x]P)\sigma, \mathtt{fn}(\nu y[y]Q)\sigma)$. Because σ and $\sigma\{w/x\}$ agree on $\mathtt{fn}(\nu x[x]P)$, so by Lemma 2.3 $(\nu x[x]P)\sigma \equiv (\nu x[x]P)\sigma\{w/x\}$, and likewise $(\nu y[y]Q)\sigma \equiv (\nu y[y]Q)\sigma\{w/y\}$. Calculation according to lemma 2.6 gives

$$\mathtt{fn}((\nu x[x]P)\sigma\{w/x\}, (\nu y[y]Q)\sigma\{w/y\}) = \mathtt{fn}(P\sigma\{w/x\}, Q\sigma\{w/y\}) - \{w\}$$

Thus we have $z\sigma \notin \mathtt{fn}(P\sigma\{w/x\}, Q\iota\{w/y\}) - \{w\}$. Because we assume $w \neq z\sigma$, so $z\sigma \notin \mathtt{fn}(P\sigma\{w/x\}, Q\iota\{w/y\})$, then applying lemma 2.16 on the result of the second step we get

$$(P\sigma\{w/x\})\iota\{z\sigma/w\} \dot{\sim} (Q\sigma\{w/y\})\iota\{z\sigma/w\}$$

It is clear that $w \notin \{v\sigma \mid v \in (\mathtt{fn}(P) - \{x\}\}$ and $w \notin \{v\sigma \mid v \in (\mathtt{fn}(Q) - \{y\}\}$, so by corollary 2.4, $P\sigma\{z\sigma/x\} \dot{\sim} Q\sigma\{z\sigma/y\}$.

Now to finish the proof, since $\sigma\{z\sigma/x\} = \iota\{z/x\} \circ \sigma$ and $\sigma\{z\sigma/y\} = \iota\{z/y\} \circ \sigma$, thus $(P\iota\{z/x\})\sigma \dot{\sim} (Q\iota\{z/y\})\sigma$. $\square$

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 3. 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 3 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 3, 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

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

suppose the premisses have the property, that is to say $P\iota \succ x\iota.F\iota$ and $Q\iota \succ \bar{y}\iota.C\iota$, we need to show that the conclusion also has the property. From $P\iota \succ x.F\iota$ and $Q\iota \succ \bar{y}.C\iota$, by Lemma 2.14, it follows that

$$P\iota\{x/y\} \succ x\iota\{x/y\}.F\iota\{x/y\} \text{ and } Q\iota\{x/y\} \succ \bar{y}\iota\{x/y\}.C\iota\{x/y\}$$

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 $$\frac{P \,|\, !P \succ_\sigma \alpha.A}{!P \succ_\sigma \alpha.A}$$

Figure 3: Inference Rules for Conditional Commitments

Notice that $\bar{y}\iota\{x/y\} = \overline{x\iota\{x/y\}}$, and $(F \cdot C)\iota\{x/y\} \equiv F\iota\{x/y\} \cdot C\iota\{x/y\}$ by Lemma 2.12, it follows immediately that $(P|Q)\iota\{x/y\} \succ \tau.(F \cdot C)\iota\{x/y\}$.

For C-Rest, suppose the premis has the property, that is to say $P\sigma \succ \alpha\sigma.A\sigma$, we will show that the conclusion also has the property, that is $(\nu x P)\sigma \succ \alpha\sigma.(\nu x A)\sigma$. So we start from $P\sigma \succ \alpha\sigma.A\sigma$. Let $z = \texttt{fresh}(\texttt{fn}(P\sigma))$. By Lemma 2.14, $(P\sigma)\iota\{z/x\} \succ (\alpha\sigma)\iota\{z/x\}.(A\sigma)\iota\{z/x\}$. Because σ is x clean, $\sigma \circ \iota\{z/x\} = \sigma\{z/x\}$. Thus $P\sigma\{z/x\} \succ \alpha\sigma\{z/x\}.A\sigma\{z/x\}$. Since $x \notin name(\alpha)$, $\alpha\sigma\{z/x\} = \alpha\sigma$. Moreover, by Lemma 2.13, $name(\alpha\sigma) \subseteq \texttt{fn}(P\sigma)$, so $z \notin name(\alpha\sigma)$. Then apply Rest $\nu z P\sigma\{z/x\} \succ \alpha\sigma.(\nu z A\sigma\{z/x\})\iota$. That is $(\nu x P)\sigma \succ \alpha\sigma.(\nu x A)\sigma$, because $(\nu x P)\sigma \equiv \nu z P\sigma\{z/x\}$, and it is not difficult to see that $(\nu x A)\sigma \equiv (\nu z A\sigma\{z/x\})\iota$. □

The following lemma shows that conditional commitments provide us with a means to analyze commitments of $P\sigma$.

Lemma 3.3 *For a process P and substitution σ, $P\sigma \succ \alpha.A$ if and only if $P \succ_\rho \beta.A'$*

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

Proof The direction "if" follows directly from Lemma 3.2 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. $Q\sigma \succ \alpha.B, A \equiv P\sigma|B$.
3. $P\sigma \succ x.P', Q\sigma \succ \bar{x}.Q', A \equiv P'|Q'$.
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, $P \succ_\rho \beta.F'$ with $x = \beta\sigma, F \equiv F'\sigma$, and for some ρ', $\sigma = \rho\circ\rho'$. In this case $\beta = y$ for some name y, otherwise $\beta\sigma = \tau$. So by Lemma 3.1 $\rho = \iota$, and thus $\rho' = \sigma$. For the same reason $Q \succ_\iota \bar{z}.C'$ with $x = z\sigma$, $C \equiv C'\sigma$. It is clear that F', C' must be abstraction and concretion respectively, thus by C-Com $P|Q \succ_{\iota\{y/z\}} \tau.(F' \cdot C')$. Moreover in this case $F \cdot C \equiv (F' \cdot C')\sigma$ and $\sigma = \iota\{y/z\} \circ \sigma$.

Other cases can be proved similarly. □

Now let $P \sim_D Q$, how can we characterize this operationally in terms of the commitments of P and Q? More specifically, if $P \succ \alpha.A$ for some A, what Q should be able to do in order to have the relation $P \sim_D Q$? Follow the definition of bisimulation, one may want to say the following:

(1) Whenever $P \succ \alpha.A$, then $Q \succ \alpha.B$ for some B such that $A \sim_D B\ldots$

However this is too strong for D–bisimulation. Let us look at the following two processes P and Q

$$\begin{array}{rcl} P & \equiv & \alpha.(x|\bar{y}) + \alpha.(x.\bar{y} + \bar{y}.x) + \alpha.(x|\bar{x}) \\ Q & \equiv & \alpha.(x.\bar{y} + \bar{y}.x) + \alpha.(x|\bar{x}) \end{array}$$

It is easy to see that $P \sim Q$. But for $P \succ \alpha.(x|\bar{y})$ there is no Q' such that $Q \succ \alpha.Q'$ and $Q' \sim x|\bar{y}$. A closer look reveals that in this case $P \succ \alpha.(x|\bar{y})$ is actually matched by different commitments of Q according to whether x and y are the same name. Since $x|\bar{y} \sim_{\{x,y\}} x.\bar{y}+\bar{y}.x$, if $x \neq y$ then $P \succ \alpha.(x|\bar{y})$ can be matched by $Q \succ \alpha.(x.\bar{y} + \bar{y}.x)$. If $x = y$, then it can be matched by $Q \succ \alpha.(x|\bar{x})$. Thus Q

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 is a (assumed) distinction and $\mathcal{S}$ includes some (presupposed, or to be established) D-bisimulation relation. Thus in the case of P and Q here we will write $Q \succ_{\mathcal{S}}^{\emptyset} (x|\bar{y})$, where $\mathcal{S} = \{x|\bar{y} \sim_{\{x,y\}} x.\bar{y} + \bar{y}.x, x|\bar{x} \sim x|\bar{x}\}$ and $\emptyset$ is the empty distinction. Then we give the characterization of $\sim_D$ with the help of $\succ_{\mathcal{S}}^{D}$.

Definition 3.4 *Let $\mathcal{S}$ be a set of triples of the form (A, D, B) where A, B are agents, D is a distinction. Then $\succ_{\mathcal{S}}$ is the smallest set of triples of process, distinction, and commitment such that for $(P, D, \alpha.A) \in \succ_{\mathcal{S}}$, which we will write $P \succ_{\mathcal{S}}^{D} \alpha.A$ from now on, then P is a process, D is a distinction, and $\alpha.A$ is a commitment, and moreover, one of the following must holds:*

1. *$P \succ \alpha.B$ for some B with $(A, D, B) \in \mathcal{S}$, or*

2. *P has two free names x, y such that $(x, y) \notin D$ and $P \succ_{\mathcal{S}}^{D\cup\{x,y\}} \alpha.A$ and $P\iota\{y/x\} \succ_{\mathcal{S}}^{D\iota\{y/x\}} \alpha\iota\{y/x\}.A\iota\{y/x\}$.*

Lemma 3.5 *If $P \succ_{\mathcal{S}}^{D} \alpha.A$, then for every $\sigma \models D$ there exist D', B such that $P\sigma \succ \alpha\sigma.B\sigma, \sigma \models D'$ and $(A, D', B) \in \mathcal{S}$.*

Proof This is proved by induction on the number of pairs $(x, y) \notin D$ such that x, y are different free names of P. If there is no such pair, then by the above definition it must be the case that $P \succ \alpha.B$ with $(A, D, B) \in \mathcal{S}$. Thus $P\sigma \succ \alpha\sigma.B\sigma$ and the fact holds by taking D' to be D. If there is $(x, y) \notin D$ with $x, y \in \mathtt{fn}(P)$, then either we have $P \succ \alpha.B$ with $(A, D, B) \in \mathcal{S}$ as above, or by the definition of $\succ_{\mathcal{S}}$ we have $P \succ_{\mathcal{S}}^{D\cup\{x,y\}} \alpha.A$ and $P\iota\{y/x\} \succ_{\mathcal{S}}^{D\iota\{y/x\}} \alpha\iota\{y/x\}.A\iota\{y/x\}$. Now if $x\sigma \neq y\sigma$, then $\sigma \models D\cup\{x, y\}$, and because $P \succ_{\mathcal{S}}^{D\cup\{x,y\}} \alpha.A$ by the induction hypothesis there exists D', B such that $P\sigma \succ \alpha\sigma.B\sigma, \sigma \models D'$ and $(A, D', B) \in \mathcal{S}$. When $x\sigma = y\sigma$, we use $P\iota\{y/x\} \succ_{\mathcal{S}}^{D\iota\{y/x\}} \alpha\iota\{y/x\}.A\iota\{y/x\}$ and the argument is similar. □

The relation $\succ_{\mathcal{S}}$ is not very easy to understand by its definition. The following theorem characterizes it in the important case when $\mathcal{S}$ is the largest distinction bisimulation.

Theorem 3.6 *When $\mathcal{S} = \{(A, D, B) \mid A \sim_D B\}$, the following conditions are equivalent*

1. *$P \succ_{\mathcal{S}}^{D} \alpha.A$*

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

Proof $1 \Rightarrow 2$: Suppose $P \succ_{\mathcal{S}}^{D} \alpha.A$. If $\sigma \models D$, according to Lemma 3.5, there exist D', B such that $P\sigma \succ \alpha\sigma.B\sigma, \sigma \models D'$ and $A \sim_{D'} B$, thus $A\sigma \dot{\sim} B\sigma$. So whenever $\sigma \models D$ there exists B' such that $P\sigma \succ \alpha\sigma.B'$ and $B' \dot{\sim} A\sigma$.

$2 \Rightarrow 1$: We prove this by induction on the number of pairs $(x, y) \notin D$ such that $x \neq y$ and $x, y \in \mathtt{fn}(A, P)$. The base case is that there is no such pair of names, 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 $P \succ_{\mathcal{S}}^{D} \alpha.A$.

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 $\succ_{\mathcal{S}}$, $P \succ_{\mathcal{S}}^{D} \alpha.A$. $\square$

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 3.6 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 3.6, 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
3. $P \succ_{\sigma} \alpha.A'$ implies $P\sigma \succ \alpha\sigma.A'\sigma$ by Lemma 3.2, which implies $P\sigma\rho \succ \alpha\sigma\rho.A'\sigma\rho$ by Lemma 2.14.

Now consider the case $\texttt{norma}(A) \equiv (x)P, \texttt{norma}(B) \equiv (y)Q$. We will show that in this case there exists $z \notin \texttt{fn}(A, B)$ such that

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

By Theorem 2.19 $A \sim_D B$ implies $(x)P \sim_D (y)Q$, thus by proposition 2.24, $P\iota\{z/x\} \sim_{D\backslash z} Q\iota\{z/y\}$ for any $z \notin \texttt{fn}(A, B)$. Thus take any $z \notin \texttt{fn}(A, B)$ (such z certainly does exist) we have $(P\iota\{z/x\}, D\backslash z, Q\iota\{z/y\}) \in \mathcal{S}$.

Next for the case $\texttt{normc}(A) \equiv [x]P, \texttt{normc}(B) \equiv [x]Q$, we will show

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

Again by Theorem 2.19 $[x]P \sim_D [x]Q$. Thus the above follows directly from proposition 2.23.

Now consider the case $\texttt{normc}(A) \equiv \nu x[x]P, \texttt{normc}(B) \equiv \nu y[y]Q$. We will show that there exists $z \notin \texttt{fn}(A, B)$ such that

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

where $D' = \{z\} \times \texttt{fn}(A, B) \cup \texttt{fn}(A, B) \times \{z\}$. Because in this case $\nu x[x]P \sim_D \nu y[y]Q$, and by proposition 2.25 $P\iota\{z/x\} \sim_{D \cup D'} Q\iota\{z/y\}$ for any $z \notin \texttt{fn}(A, B)$. Thus such z can be found. □

Theorem 3.9 *If A and B are symbolicly D–bisimilar, then they are D–bisimilar.*

Proof Suppose $\mathcal{S}$ is a symbolic bisimulation. We will show that

$$\mathcal{B} = \{(A\sigma, B\sigma) \mid (A, D, B) \in \mathcal{S}, \sigma \models D\}$$

is a simulation. It is exactly the same to show that $\mathcal{B}^{-}$ is also a simulation. So it follows that $\mathcal{B}$ is a bisimulation.

Suppose $(A\sigma, B\sigma) \in \mathcal{B}$ with D as witness, i.e. $(A, D, B) \in \mathcal{S}$ and $\sigma \models D$. We have to cover the following cases.

If (A, B) is a pair of processes (P, Q), then $(A\sigma, B\sigma)$ is a pair of processes of the form $(P\sigma, Q\sigma)$. Suppose $P\sigma \succ \alpha.A'$, by lemma 3.3 $\sigma = \rho \circ \rho'$ such that $P \succ_{\rho} \beta.A''$ for some ρ, β, A'' with $\alpha = \beta\sigma$ and $A' \equiv A''\sigma$. Because $(P, D, Q) \in \mathcal{S}$, to match

$P \succ_\rho \beta.A''$, 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 3.5, 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 $u = \mathtt{fresh}(\{z\sigma \,|\, z \in \mathtt{fn}(P) - \{x\}\})$ and $v = \mathtt{fresh}(\{z\sigma \,|\, z \in \mathtt{fn}(Q) - \{y\}\})$. 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 $u = \mathtt{fresh}(\{z\sigma \,|\, z \in \mathtt{fn}(P) - \{x\}\})$ and $v = \mathtt{fresh}(\{z\sigma \,|\, z \in \mathtt{fn}(Q) - \{y\}\})$. 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 $((P\iota\{z/x\})\sigma\{w/z\}, (Q\iota\{z/y\})\sigma\{w/z\}) \in \mathcal{S}$. $\square$

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'". Then in order to match a symbolic transition, a finite collection of boolean expression B is to be found such that $\vee B = b$ for some boolean b. However, there is not an effective method in general to find such B. In this paper, due to the simplicity of dealing with names which is a very simple form of value, similar thing is achieved by introducing the relation $\succ^D_S$ which is constructively defined.

In [Ama92], Amadio introduced *uniform strong bisimulation* for π-calculus. The definition of uniform strong bisimulation is very similar to the definition of symbolic D-bisimulation in this paper without the more involved part of $\succ^D_S$. The result is that the resulting equivalence get is strictly stronger than the bisimulation congruence.

In [San93], Sangiorgi introduced the notion of *open bisimulation*. The equivalence resulting from open bisimulation is a congruence for π-calculus, and it is strictly stronger than the bisimulation congruence in this paper. He then gave an effective characterization for open bisimulation. His characterization is very similar to the characterization of D-bisimulation by symbolic D-bisimulation, except that the complication of $\succ^D_S$ is not necessary for open bisimulation.

A different approach to prove bisimulation congruence is taken in [PS93], where an axiom system is presented for π–calculus processes. Unlike our approach where we work with all D–bisimulation equivalences at the same time, they stay within the bisimulation congruence by extending the conditional expression with negation and disjunction. This allows any general condition like *true* to be split into a logically equivalent disjunctive formula ψ, which achieves the effect similar to $\succ^D_S$.

The work carried out in the paper focuses on the the congruence induced by the *late* version of bisimulation equivalence. It would be interesting to see whether a similar characterization can be found for the early version. This is not obvious since in the symbolic definition is very close to that of late bisimulation. Also we leave out the match construction $[x = y]P$ of the π–calculus of [MPW92a]. The match construction does not give the π–calculus extra power. The theory developed here can cope with inclusion of this construction by introducing compound boolean expression in the conditional commitment. These could be a topic for further research.

Acknowledgements The author would like to thank Matthew Hennessy for reading a draft of the paper and for many insiteful comments and suggestions. The author would also like to thank Alan Jeffrey and Huimin Lin for inspiring discussions on the topic of the paper.

References

[Ama92] R. Amadio. A uniform presentation of chocs and π–calculus. Technical Report Rapport de recherche 1726, INRIA-Lorraine, Nancy, 1992.

[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.

[PS93] J. Parrow and Davide Sangiorgi. Algebraic theories for value–passing calculi. Technical report, Department of Computer Science, University of Edinburgh, 1993. Forthcoming.

[San93] D. Sangiorgi. A theory of bisimulation for π–calculus. Technical report, University of Edinburgh, 1993. Forthcoming.

[Sto88] Allen Stoughton. Substitution revisited. *Theoretical Computer Science*, 59:317–325, 1988.