On the Expressiveness of Interaction[*]

Yuxi Fu Hao Lu

BASICS, Department of Computer Science
MOE-MS Key Laboratory
for Intelligent Computing and Intelligent Systems
Shanghai Jiao Tong University, Shanghai 200240, China

February 12, 2010

Abstract

Subbisimilarity is proposed as a general tool to classify the relative expressive power of process calculi. The expressiveness of several variants of CCS is compared in terms of the subbisimilarity relationship. Similar investigation is also carried out for the variants of the pi calculus. The relative expressiveness of the different forms of the choice operation and the different ways of introducing infinite behaviors are systematically studied in both the frameworks of CCS and pi. Some issues concerning the expressiveness of both CCS and pi are clarified. Several open problems are solved along the way. The subbisimilarity approach and the relative expressiveness results are applied to show the independence of the operators of the pi calculus. The definition of the subbisimilarity relationship can be further strengthened with computational requirement, leading to a uniform treatment of computation and interaction.

Keywords: Process calculus, expressiveness, bisimulation

Contents

1 Computation and Interaction **3**
1.1 Criteria for Expressiveness 3
1.2 Results on Expressiveness 6
1.3 Issues to be Addressed 7
1.4 Contributions 9

[*]Appeared in *Theoretical Computer Science*, **411**: 1387-1451, 2010

2 Subbisimilarity for CCS **10**
2.1 Semantic Theory . . . 10
2.2 Bisimulation Theory . . . 15
2.2.1 Subbisimilarity . . . 17
2.2.2 Subbisimilarity and Church-Turing Thesis . . . 19
2.3 Choice . . . 23
2.3.1 Termination Preserving Encoding . . . 28
2.3.2 Mixed Choice and Guarded Recursion . . . 29
2.4 First Order Recursion . . . 37
2.4.1 Expansion Property of Replication . . . 38
2.4.2 Replication and Mixed Choice . . . 43
2.4.3 Replication, or Fixpoint . . . 46
2.5 Recursive Definition . . . 47
2.5.1 Fixpoint, or Constant Definition . . . 50
2.6 Interaction Hierarchy of CCS . . . 60

3 Subbisimilarity for Pi **63**
3.1 Pi Subbisimilarity . . . 66
3.2 Choice in Pi . . . 68
3.3 Replication in Pi . . . 71
3.4 Parametric Definition in Pi . . . 72
3.5 Interaction Hierarchy of Pi . . . 77

4 Application of Subbisimilarity **79**
4.1 Operator Independence . . . 79
4.2 Model Assessment . . . 82
4.2.1 CCS with Sufficient Computing Power . . . 83
4.2.2 Pi with Simple Semantics . . . 83

5 More Subbisimilarities **85**
5.1 Name Escape Communication . . . 85
5.2 Value-Passing CCS . . . 86
5.3 Asynchronous Pi . . . 88
5.4 Internal Communication . . . 93

6 Effective Subbisimilarity **94**

7 Towards a Model Independent Theory **98**

A Proof of Proposition 10 **100**

1 Computation and Interaction

A fundamental issue in process theory is the expressiveness of process calculi. The most crucial step in studying the expressiveness is to pin down a reasonable definition of expressiveness. What does it mean that one process calculus is (strictly) more expressive than another? We start with an overview of some of the criteria used in the literature and some expressiveness results concerning the π-calculus [43].

1.1 Criteria for Expressiveness

One may start looking for the expressiveness criteria by asking the following question: what is the essential difference between the expressiveness of the process calculi and the expressiveness of the Turing computing models [58]? Process calculi are operational models of *open* computing systems that formalize interaction between systems. In this sense process theory is a theory of interaction [41]. Turing computing models like λ-calculus [5] study *closed* computing systems whose only interactions with environments are input and output actions. Two computable functions are compared by their impacts on the environments, which are but their input-output behaviors. In other words, the input and the output of a computable function are the only interactions of it with environments. The extensional functional equality amounts to saying that a function is completely determined by the impacts of these interactions. Generalizing this view, a process is identified by the effect it may inflict on environments. Since processes normally interact with environments continuously, the behaviors of processes have to be judged in a co-inductive way. The bisimulation approach was introduced precisely for this purpose [39, 50, 57].

The above discussion leads to an interactional approach to the study of process expressiveness. The expressive powers of process calculi are compared in terms of their interactabilities, the ability to interact with environments. In process theory, interactabilities are defined using labeled transition systems. Now let the operational semantics of $\mathcal{L}_1$ and $\mathcal{L}_2$ be defined respectively by the labeled transition systems $(\mathcal{S}_1, \mathcal{T}_1, \longrightarrow_1)$ and $(\mathcal{S}_2, \mathcal{T}_2, \longrightarrow_2)$. Let $\asymp_1$ and $\asymp_2$ be some chosen observational equivalences of $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. Suppose $\langle [\![\_]\!], \widehat{\_} \rangle$ is a translation from $\mathcal{L}_1$ to $\mathcal{L}_2$, where $[\![\_]\!]$ is a map from $\mathcal{S}_1$ to $\mathcal{S}_2$ and $\widehat{\_}$ is a map from $\mathcal{T}_1$ to $\mathcal{T}_2$. The following criteria have been used to judge the quality of the translation [49, 46].

1. *Operational Completeness* is, from the point of view of interaction, the minimal criterion for "being more expressive". Process calculi are operational models. For $\mathcal{L}_2$ to be at least as expressive as $\mathcal{L}_1$, the former must be able to simulate the operational behaviors of the latter. This normally means that $[\![P_1]\!] \overset{\widehat{\lambda}}{\Longrightarrow}_2 \asymp_2 [\![P_2]\!]$ whenever $P_1 \xrightarrow{\lambda}_1 P_2$. From the viewpoint of interactions, Operational Completeness alone is just not strong enough. It may well be that $[\![P]\!]$ can engage in an action to which P does not have any counterpart action.

2. *Operational Soundness* requires that whatever $[\![P]\!]$ can do is the translation of an action of P. Formally it requires that $[\![P]\!] \stackrel{\lambda}{\Longrightarrow}_2 P_2$ always implies that $P \stackrel{\lambda'}{\Longrightarrow}_1 P_1$ for some λ', P_1 such that $[\![P_1]\!] \asymp_2 P_2$ and $\widehat{\lambda'} = \lambda$. Operational Soundness guarantees that $[\![P]\!]$ cannot affect the environments more than P via interactions.

3. *Operational Correspondence* consists of Operational Completeness and Operational Soundness. This is a basic criterion when comparing the interactional behaviors of process calculi. In the cases when $\mathcal{L}_1, \mathcal{L}_2$ are variants to each other and $\asymp_1, \asymp_2$ are bisimulation-like equivalences, Operational Correspondence sometimes implies Observational Correspondence.

4. *Observational Completeness* is the minimal criterion from the point of view of observation. Formally it imposes the condition that $P \asymp_1 Q$ whenever $[\![P]\!] \asymp_2 [\![Q]\!]$. Observational Completeness often comes for free. Since all the contexts of $\mathcal{L}_1$ are translated to the contexts of $\mathcal{L}_2$, it follows that P, Q should be indistinguishable if their translations in $\mathcal{L}_2$ are indistinguishable.

5. *Observational Soundness* places a substantial demand on the quality of the translation. It says that $[\![P]\!] \asymp_2 [\![Q]\!]$ whenever $P \asymp_1 Q$. Being a reasonable criterion from the point of view of observation, Observational Soundness is difficult technically. Many encodings studied in the literature fail to meet this condition.

6. *Observational Correspondence*, often called *Full Abstraction*, is the combination of Observational Completeness and Observational Soundness. The advantage of Full Abstraction is its wide applications. Like Operational Correspondence, it can be applied to compare two process calculi with quite different action sets, say π-calculus [43] and the ambient calculus [12]. It must be said however that Full Abstraction is not of much use without Operational Correspondence.

7. *Computation Criterion* asks that a process that might engage in an infinite computation is distinguished from a process that cannot do that. How can this computational property be used to classify interactions? Well in a theory of interaction, computations are internal interactions of systems. If a process is exclusively engaged in an infinite computation, the process will never get a chance to interact externally. So computations may interfere with interactions by not giving the latter any chance! Let's say that P is terminating if P cannot perform any infinite sequence of τ-actions. In this paper Computation Criterion refers to the following property.

   > For every process P of $\mathcal{L}_1$, P is terminating if and only if $[\![P]\!]$ is terminating.

   Computation Criterion has been used in the previous work [49, 7, 8]. It can be seen as a reincarnation, in a general theory of interaction, of one basic understanding of the Turing models, where a function that delivers an output upon receiving an input is different from any function that does not deliver anything at all upon receiving the same input.

A different approach to the study of expressiveness in process calculi, the reductional approach, is based on the reductional semantics [34, 40]. In fact the reductional approach has been more popular among the two. In this approach only computations are bisimulated. In other words, it is based on the Operational Correspondence restricted to computation. The bisimulation of computation alone is not a strong enough property for the models of interactions. Additional properties are necessary to uplift the reductional approach. The solutions that have been proposed are to impose some additional structural conditions and/or success-testing properties. Let's take a look at some of them.

- *Homomorphism* refers to the property that $[\![P \,|\, Q]\!] \asymp_2 [\![P]\!] \,|\, [\![Q]\!]$ for all processes P, Q of $\mathcal{L}_1$ [49]. If we think of it, what this condition really says is that the concurrent composition operator is model-independent.
- *Stability* [49] is not as popular as Homomorphism. It can be formulated as $[\![P\sigma]\!] \asymp_2 [\![P]\!]\sigma$ for every process P of $\mathcal{L}_1$ and every substitution σ. Stability is based on the idea that the names are propertyless. If there is a translation from $\mathcal{L}_1$ to $\mathcal{L}_2$, there should be a translation from $\mathcal{L}_1$ to $\mathcal{L}_2$ that is parametric on the names.
- *Compositionality* [28, 29] is a generalization of Homomorphism. It postulates that the translations of all the operators of $\mathcal{L}_1$ are structural.
- *Barbedness* [45] is a special success-testing property. It requires that if a process is capable of performing a certain type of observable actions, so is its translation; and vice versa. Barbedness offers a simple and versatile condition for the reductional approach.

Palamidessi uses *Uniformity* and *Reasonability* as criteria to compare the expressiveness of the π-calculi [49]. Uniformity consists essentially of stronger versions of Homomorphism and Stability where the equivalence $\asymp_2$ is replaced by the syntactical congruence $\equiv$. Reasonability is related to Barbedness. The following is taken from [49].

> Concerning the notion of semantics, we call "reasonable" a semantics which distinguishes two processes P and Q whenever there exists a maximal (finite or infinite) computation of P in which the intended observables (some visible actions) are different from the observables in any (maximal) computation of Q.

Uniformity and Reasonability make it easy to differentiate various process calculi using the leader electoral systems [49, 51, 62]. Significant results using the reductional approach have also been reported in a number of papers by Gorla. See for example [28, 29, 30, 31]. The criteria Gorla has applied to analyze several groups of process calculi are Compositionality, Name Invariance (a variant of Stability), Operational Correspondence, Computation Criterion, and

$$\begin{tikzcd}
 & & & \pi \arrow[dl, dashed, "?"] \\
 & & \pi^m \arrow[ur, hook] \arrow[dl, "/" marking] & \\
 & \pi^s \arrow[ur, hook] \arrow[dd] & & \\
\pi^i \arrow[ur, hook] \arrow[dr] & & \pi^- \arrow[ul, hook] \arrow[dl] & \\
 & \pi^a \arrow[ul, hook] \arrow[ur, hook] & &
\end{tikzcd}$$

Figure 1: Encodings Between Pi Variants

Barbedness. The combined force of these criteria is duly demonstrated in the separation results in *loc. cit.*

In [7, 8], Busi, Gabbrielli and Zavattaro made use of decidability properties to tell apart process calculi. The weak point of the decidability criterion is that two formulations of an operational semantics might have different decidability properties on the one hand and the same interactability on the other. Although decidability *per se* does not say anything about interactability, it does often imply the absence of Turing completeness, which is a strong separation criterion. Moreover the techniques used to establish decidability results can often be modified to prove separation results.

1.2 Results on Expressiveness

We now sketch some of the relative expressiveness results about the variants of π-calculus. Let π be π-calculus with the binary choice (for example $P + Q$), π^m the π with the mixed choice (for example $a(x).P + \overline{b}c.Q$), π^s the π with the separated choice (for example $a(x).P + b(y).Q$ and $\overline{c}d.P + \overline{e}f.Q$), π^i the π with the input choice (for example $a(x).P + b(y).Q$), π^- the π without the choice, and π^a the asynchronous π. Palamidessi summarizes in [49] the existing relative expressiveness of these variants by the diagram in Figure 1. The sub-calculus relationship is indicated by the hooked arrow $\hookrightarrow$. The plain arrow $\rightarrow$ indicates the existence of a uniform and reasonable encoding. Boudol, one of the proposers of π^a, gives in [6] a translation from π^- to π^a that preserves a Morris style contextual equivalence. Honda and Tokoro provide a different encoding whose correctness is proved for some weak bisimilarity [34]. Their asynchronous bisimilarity is adopted as the standard bisimulation equivalence for the asynchronous calculi. Nestmann and Pierce study in [46] two encodings from π^i to π^a. One is fully abstract with respect to the asynchronous bisimilarity but not termination preserving. The other is termination preserving but not fully abstract. In π^a [47] Nestmann examines some encodings of π^s and argues without a proof that no fully abstract translation from the former to the latter exists. Palamidessi proves in [49] that there is no uniform and reasonable encoding from π^m to π^s.

Some other variants of the asynchronous π-calculus have been considered in literature. In [53] the authors study the relative expressiveness of three restrictions of π^a. The $PO\pi$ calculus is obtained from π^a by forcing all the output processes to be of the form $!\overline{a}n$. In other words, the output processes are persistent. Similarly $PI\pi$ is the calculus in which all the input processes are of the form $!a(x).T$; and $P\pi$ is the variant in which both output and input processes are persistent. It is argued in [53] that $PO\pi$ and $PI\pi$ are as expressive as π^a in a certain sense, and that $P\pi$ is strictly less expressive than π^a. However the authors of [10] demonstrate that there are no encodings from π^a to any three of the variants that preserve testing equivalence [14, 32].

Some expressiveness results about Ambient Calculi are reported [42, 51, 30]. However the overall picture is not yet clear.

1.3 Issues to be Addressed

How forceful are the results stated in Figure 1? It is fruitful to take a look at these results using the criteria discussed above.

1. From the viewpoint of Operational Correspondence, the encodings indicated by the arrows of Figure 1 vary considerably. For example there is an encoding from π^i to π^a that satisfies Operational Correspondence [46]. On the other hand the known encoding from π^s to π^a satisfies much weaker properties than Operational Correspondence [47]. Using simple argument, as we shall see later, one can show that the 'tossing-the-coin algorithm', which can be programmed in π^s by $\overline{c}f + \overline{c}t$, cannot be coded up in any of π^i, π^a and π^- in a way that Operational Correspondence is respected.

2. From the point of view of Observational Correspondence, the encodings indicated by the arrows of Figure 1 are different. For instance, the nonexistence of a uniform and reasonable encoding from π^m to π^s does not rule out the existence of a fully abstract encoding from π^m to π^s, with respect to the weak bisimilarity, that respects Operational Correspondence. On the other hand the arrow from π^m to π^a, as indicated in Figure 1, is supported by an encoding much weaker than a fully abstract encoding with respect to the weak bisimilarity. Moreover, different encodings are making use of different equivalences. For the encoding $\pi^i \rightarrow \pi^a$ the asynchronous equivalence of π^i is used. In the embedding $\pi^i \hookrightarrow \pi^s$ the weak bisimilarity of π^i is used. The question of how we should relate the asynchronous calculi to the synchronous ones also comes up.

3. From the point of view of Computation Criterion, the encodings indicated by the arrows of Figure 1 are not all consistent. On the one hand, Palamidessi has essentially proved that there is no uniform and reasonable encoding from π^m to π^s that satisfies Computation Criterion, which is indicated as a negative result in Figure 1. On the other hand, an encoding of π^s in π^a, indicated as a positive result in Figure 1, can hardly satisfy Computation Criterion.

According to the above analysis, there is a real need to clarify the picture about the relative expressiveness of the π-variants. It would be fruitful to look at the models from the point of view of interactability. The computational factor should also be considered since they interfere with the interactions. In other words we expect to strengthen the results stated in Figure 1 by answering the following question.

> Question 1: What are the relationships between the π variants if Operational Correspondence and Observational Correspondence are adopted? What if Computation Criterion is also adopted?

To answer the above question, some open problems need be solved. The relative expressiveness of π over π^m has stood unanswered for quite some time. From the viewpoint of interactions, the problem can be formalized as follows: Is there a (termination preserving) encoding $[\![\_]\!]$ from π to π^m such that $P \approx [\![P]\!]$? The answer is not just practically interesting, it is also theoretically important. Similarly the relative expressiveness of π^m over π^s is not completely settled, despite of the result in [49]. The question to be answered is this: Is there a (termination preserving) encoding $[\![\_]\!]$ from π^m to π^s such that $P \approx [\![P]\!]$? There is also a question on whether any form of the external choice is redundant. In other words, is there an encoding $[\![\_]\!]$ from π^s to π^- such that $P \approx [\![P]\!]$? The choice operator is often used in specification [39], whereas the concurrent composition "$|$" is an implementation operator. The third question can be equivalently stated as follows: is there a specification that cannot be implemented without the choice operator? Most of the above problems are open even for CCS.

The π-calculus has some well known relatives. These include the value-passing CCS, the asynchronous π-calculus, the πI-calculus. It is interesting to investigate the expressiveness of these calculi using similar criteria.

> Question 2: What are the comparative expressiveness of the well known π relatives if Operational Correspondence and Observational Correspondence are adopted? What if Computation Criterion is also adopted?

Sometimes two variants only differ in the ways the infinite behaviors are introduced. The well known recursion mechanisms are the fixpoint operations, recursive definitions and replications. It is also interesting to see if different recursion mechanisms make a difference in interaction.

> Question 3: What are the comparative expressiveness of the various recursion mechanisms from the viewpoint of interaction?

Answers to the above questions will not only improve our understanding of Figure 1, but also help us to look for a model-independent theory of expressiveness. It is our belief that the theory of expressiveness is the most fundamental part of the process theory, playing a role similar to that of the theory of computability in computation theory. It is also our belief that a model-independent theory of expressiveness provides a unifying framework to integrate the process theory and the computation theory.

1.4 Contributions

We set out to answer the questions and solve the problems stated in Section 1.3. Our contributions are as follows.

1. We propose a general framework to compare the expressive powers of CCS variants and π variants in a uniform manner. The major contributions can be summarized in four statements.
    (a) Subbisimilarities are introduced as a uniform tool to study the expressiveness of interaction, incorporating both Operational Correspondence and Observational Correspondence.
    (b) The relative expressive powers of nine variants of CCS, and the corresponding variants of π-calculus, are characterized in terms of the subbisimilarity.
    (c) The relative expressive powers of these variants are also characterized in terms of the codivergent subbisimilarity in which Computation Criterion is incorporated.
    (d) The effectiveness of the subbisimilarities is formalized and its link to Church-Turing Thesis is pointed out.
2. By applying the subbisimilarity tool, we contribute to the theory of expressiveness in three aspects.
    (a) A number of relative expressiveness results are obtained. The results are concerned with the dynamic binding mechanism, value-passing mechanism, and several variations of the name-passing mechanism.
    (b) It is formally established that all the standard operators of the π-calculus are independent of each other.
    (c) It is pointed out how the expressiveness results can be used to assess the process calculi.

At a more technical level, our contributions are proof techniques for establishing the negative results about expressiveness.

The paper is organized as follows. The subbisimilarities for CCS variants are introduced in Section 2 to study the expressiveness of the choice operators and the recursions of different kind. Some open problems about CCS are solved, and some issues are clarified. The approach is extended to the setting of the π-calculus in Section 3 to investigate the relative expressiveness of the variants of π-calculus. Section 4 gives two applications of the expressiveness theory. One is about the independence of process operators. The other is about model assessment. Section 5 takes a look at the expressiveness of some π-related calculi and the issue of static binding versus dynamic binding. Section 6 adds weight to Computation Criterion by discussing the effectiveness of the subbisimilarities. Section 7 concludes with the remarks on the significance of the subbisimilarity approach and its implication to a model-independent theory of interaction. Some of the long proofs are placed in the appendix.

2 Subbisimilarity for CCS

The importance of CCS [39] cannot be over emphasized. It is a cornerstone in theory of interaction [41]. Being one of the first process calculi, CCS provides a framework in which one could study interactions in a pure and distilled form.

As an operational model, CCS is a testbed for studying expressiveness of process calculi. However discussion on the expressiveness of variants of CCS have been rare. This is unfortunate since expressiveness has to be an important issue in process theory.

In this section we shall reveal the relationships among several variants of CCS in terms of expressiveness. Apart from serving as a technical preamble, the results in this section are also interesting on their own.

2.1 Semantic Theory

The fundamental idea of theory of interaction is that a computing agent possesses a number of *interfaces* and that two computing agents interact through the interfaces. To indicate that an agent has a particular kind of interface we assign a *name* to an interface. Two agent may interact if they have interfaces of same (or complementary) names. We shall assume that there is a set of names $\mathcal{N}$ ranged over by a, b, c, d, e, f, g, h. The notation $\overline{\mathcal{N}}$ stands for $\{\overline{a} \mid a \in \mathcal{N}\}$, the set of *conames*. The set $\mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ of the *action labels* is ranged over by λ. The notation $\overline{\lambda}$ is defined as follows:

$$\overline{\lambda} \stackrel{\text{def}}{=} \begin{cases} \overline{a} & \text{if } \lambda = a, \\ a & \text{if } \lambda = \overline{a}, \\ \tau & \text{if } \lambda = \tau. \end{cases}$$

The syntax of CCS expression is defined by the following grammar:

$$E \quad := \quad \mathbf{0} \mid X \mid \lambda.E \mid E \,|\, E' \mid (a)E \mid E{+}E' \mid \mu X.E.$$

We have left out the relabeling operation for the following reason. Our interest in CCS is to obtain results that can be easily transferred to π-calculus. For that purpose the relabeling operation is not necessary. In the localization form $(a)E$, the name a is *local*. A name is *global* if it is not local. The notation $gn(\_)$ $(ln(\_))$ stands for a function that returns the set of global names (local names). In the fixpoint expression $\mu X.E$ the process variable X is *bound*. A process variable is *free* if it is not bound. The notation $FV(\_)$ $(BV(\_))$ stands for a function that returns the set of free variables (bound variables). To simplify notations, we assume that the restriction, prefix and fixpoint operators bind tighter than the composition operator, which in turn binds tighter than the choice operator. Unless otherwise indicated, the α-conversion applies to both local names and bound variables.

A CCS expression that does not contain any free variable is called a CCS process. For a process calculus $\mathcal{L}$, we shall write $\mathcal{E}_{\mathcal{L}}$ for the set of the expressions of $\mathcal{L}$ and $\mathcal{P}_{\mathcal{L}}$ for the set of the processes of $\mathcal{L}$.

We will use the standard notations and conventions in process calculi. We omit the inactive process $\mathbf{0}$ in most occasions. For instance the process expressions $A \,|\, \mathbf{0}$ and $a.b.\mathbf{0}$ are abbreviated to A and $a.b$. A finite sequence of names $a_1, \ldots, a_n$ is often abbreviated to $\widetilde{a}$. When no ambiguity arises, $\widetilde{a}$ is also used for the finite set $\{a_1, \ldots, a_n\}$. The notation $\widetilde{\_}$ will also be applied to other syntactical entities. We shall use the notation

$$\sum_{i \in I} E_i$$

where I is a finite indexing set, for finite summation, making use of the fact that the binary choice operator is associative. Similarly we write

$$\prod_{i \in I} E_i$$

for $E_{i_1} \,|\, \ldots \,|\, E_{i_n}$ where $I = \{i_1, \ldots, i_n\}$. We also write

$$\prod_{i} E$$

for $\underbrace{E \,|\, \ldots \,|\, E}_{i \text{ times}}$.

The standard operational semantics of CCS is inductively generated by the following rules.

Prefix

$$\frac{}{\lambda.E \xrightarrow{\lambda} E}$$

Composition

$$\frac{E \xrightarrow{\lambda} E'}{E \,|\, F \xrightarrow{\lambda} E' \,|\, F} \qquad \frac{E \xrightarrow{a} E' \qquad F \xrightarrow{\overline{a}} F'}{E \,|\, F \xrightarrow{\tau} E' \,|\, F'}$$

Restriction

$$\frac{E \xrightarrow{\lambda} E' \qquad a \text{ does not appear in } \lambda}{(a)E \xrightarrow{\lambda} (a)E'}$$

Choice

$$\frac{E \xrightarrow{\lambda} E'}{E + F \xrightarrow{\lambda} E'}$$

Fixpoint

$$\frac{E\{\mu X.E/X\} \xrightarrow{\lambda} E'}{\mu X.E \xrightarrow{\lambda} E'}$$

Notice that the symmetric rules are omitted. The standard reference book on CCS [39] covers the fundamental theory about the model. We will assume that the reader is familiar with the labeled transition semantics and the basic operators in process calculi. As usual we will write $\Longrightarrow$ and $\overset{\widehat{\lambda}}{\Longrightarrow}$ for their standard interpretations. We write $E \xrightarrow{\lambda}$ if $E \xrightarrow{\lambda} E'$ for some E', and $E \not\rightarrow$ if there is no E' such that $E \xrightarrow{\tau} E'$.

A process variable X in E is *guarded* if every occurrence of X in E is underneath some prefix operation. We write α to stand for a *renaming* like $\{b_1/a_1, \ldots, b_n/a_n\}$. We write $\{E_1/X_1, \ldots, E_n/X_n\}$ for substitutions of variables. When applying the substitution to a process expression E, notation $E\{E_1/X_1, \ldots, E_n/X_n\}$, we get a process expression obtained by replacing $\widetilde{X}$ by respective $\widetilde{E}$. Bound names need be renamed to avoid name capture. For instance $(e)(f)(\overline{c}e \,|\, \overline{d}f \,|\, X)\{\overline{a}e \,|\, \overline{b}f/X\}$ is the expression $(g)(h)(\overline{c}g \,|\, \overline{d}h \,|\, \overline{a}e \,|\, \overline{b}f)$. Sometimes, especially when studying dynamic binding calculi, we need substitutions that do not admit α-conversion. For that purpose we introduce the dynamic substitution of variables $[E_1/X_1, \ldots, E_n/X_n]$. For instance the syntactic object $(e)(f)(\overline{c}e \,|\, \overline{d}f \,|\, X)[\overline{a}e \,|\, \overline{b}f/X]$ is the expression $(e)(f)(\overline{c}e \,|\, \overline{d}f \,|\, \overline{a}e \,|\, \overline{b}f)$. The substitutions of variables will be ranged over by ς. Sometimes we write $E[X_1, \ldots, X_n]$ to indicate explicitly that $X_1, \ldots, X_n$ occur in E. Accordingly we write $E[E_1, \ldots, E_n]$ for $E\{E_1/X_1, \ldots, E_n/X_n\}$ when no confusion arises. Occasionally we extend the substitutions of variables to the substitutions of expressions. We write $E\{F/\mu Z.G\}$ for instance for the expression obtained from E by replacing all the occurrences of the subexpression $\mu Z.G$ of E by F.

Lemma 1 (Stability Lemma)**.** *The following statements are valid.*
(i) If $E \xrightarrow{\lambda} E'$ *then* $E\alpha \xrightarrow{\lambda\alpha} E'\alpha$.
(ii) If $E \xrightarrow{\lambda} E'$ *then* $E\{H/X\} \xrightarrow{\lambda} E'\{H/X\}$.
(iii) Suppose $a \notin gn(E)$. *If* $E\{a/X\} \xrightarrow{\lambda} E'\{a/X\}$ *then* $E \xrightarrow{\lambda} E'$.

For $n \geq 0$, an n-step *derivative* of a process expression E is an E' such that $E \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_n} E'$. Let $\mathfrak{Drv}^n(E)$ be the set of the n-step derivatives of E. By definition $\mathfrak{Drv}^0(E) = \{E\}$. Let $\mathfrak{Drv}(E)$ and $\mathfrak{Drv}^+(E)$ be $\bigcup_{n\geq 0} \mathfrak{Drv}^n(E)$ and $\bigcup_{n\geq 1} \mathfrak{Drv}^n(E)$ respectively. A *computation* of E is either an infinite internal action sequence $E \xrightarrow{\tau} \ldots \xrightarrow{\tau} E_i \xrightarrow{\tau} \ldots$ or a finite internal action sequence $E \xrightarrow{\tau} \ldots \xrightarrow{\tau} E'$. Let $\mathfrak{Cmp}(E)$ be the set of all the computations of E. The expression E is *terminating* if $\mathfrak{Cmp}(E)$ does not contain any infinite sequence. It is *divergent* otherwise.

The complexity of the operational theory of CCS is caused by the fixpoint operator. There are very few results, if any, that characterize the infinite behaviors of the process expressions of CCS. However the subtle difference between two variants of CCS is very likely to do with the infinite behaviors. The following lemma offers an important tool to analyze the operational behaviors of the fixpoint operations.

Lemma 2 (Recursion Lemma)**.** *Suppose* $G[\mu X.E] \xrightarrow{\lambda} K$. *The following statements are valid.*
(i) $K \equiv G'[\mu X.E]$ *for some expression* $G'[X]$.

(ii) There exists some natural number i such that $G[\underbrace{E[E[\ldots[E}_{i \text{ times}}[X]]\ldots]]] \xrightarrow{\lambda} G'[X]$ *and i is no greater than the height of the derivation of* $G[\mu X.E] \xrightarrow{\lambda} K$.

Proof. The proof is a typical example of induction on the height of derivation. We take a look at two cases.

- $G \equiv G_1 \,|\, G_2$. If for example the transition $G[\mu X.E] \xrightarrow{\tau} K_1 \,|\, K_2$ is caused by $G_1[\mu X.E] \xrightarrow{a} K_1$ and $G_2[\mu X.E] \xrightarrow{\overline{a}} K_2$. By induction hypothesis some $i_1, i_2, G_1'[X], G_2'[X]$ exist such that $K_1 \equiv G_1'[\mu X.E]$, $K_2 \equiv G_2'[\mu X.E]$,
$$G_1[\underbrace{E[\ldots[E}_{i_1 \text{ times}}[X]]\ldots]] \xrightarrow{a} G_1'[X]$$
and
$$G_2[\underbrace{E[\ldots[E}_{i_2 \text{ times}}[X]]\ldots]] \xrightarrow{\overline{a}} G_2'[X].$$
Now assume $i_1 \geq i_2$. Let $G'[X]$ be $G_1'[X] \,|\, G_2'[\underbrace{E[\ldots[E}_{i_1 - i_2 \text{ times}}[X]]\ldots]]$. It is obvious that $G[\underbrace{E[\ldots[E}_{i_1 \text{ times}}[X]]\ldots]] \xrightarrow{\tau} G'[X]$.

- $G \equiv \mu Y.G_1[X, Y]$. It follows from $\mu Y.G_1[\mu X.E, Y] \xrightarrow{\lambda} K$ that
$$G_1[\mu X.E, \mu Y.G_1[\mu X.E, Y]] \xrightarrow{\lambda} K$$
is derivable with a shorter derivation. Let $G_2[X]$ be $G_1[X, \mu Y.G_1[X, Y]]$. Then by induction hypothesis one immediately sees that some $i, G'[X]$ exist such that $G'[\mu X.E] \equiv K$ and
$$G_2[\underbrace{E[\ldots[E}_{i \text{ times}}[X]]\ldots]] \xrightarrow{\lambda} G'[X].$$
It follows that $G[\underbrace{E[\ldots[E}_{i \text{ times}}[X]]\ldots]] \xrightarrow{\lambda} G'[X]$.

We are done. □

In what follows, we often need to specify a particular occurrence of a process variable. We shall write $E\langle X\rangle$ to indicate that a particular occurrence of a free process variable X in E has been specified. If for example $E \equiv a.X \,|\, B + A \,|\, X$ then the specified occurrence could be the second X. We will write for example $E\langle F/X\rangle$ for the expression obtained by replacing the specific occurrence of X by F. If $E \equiv a.X \,|\, B + A \,|\, X$ and the specified occurrence of X is the second X then $E\langle F/X\rangle$ is $a.X \,|\, B + A \,|\, F$. Using this notation we could write $E\{G/X\}\langle F/X\rangle$

for $a.G \mid B + A \mid F$. This is the expression obtained by replacing all the occurrences of X in $E\langle F/X\rangle$ by G. When no ambiguity arises we sometimes write $E\langle F\rangle$ for $E\langle F/X\rangle$. To describe the operational property of $E\langle X\rangle$, we introduce the notation $(E\langle G/X\rangle)^\dagger$ inductively defined by the following clauses.

$$\begin{aligned}
(\langle G/X\rangle)^\dagger &\stackrel{\text{def}}{=} \langle G/X\rangle, \\
(\lambda.E\langle G/X\rangle)^\dagger &\stackrel{\text{def}}{=} \lambda.E\langle G/X\rangle, \\
(E' \mid E\langle G/X\rangle)^\dagger &\stackrel{\text{def}}{=} E' \mid (E\langle G/X\rangle)^\dagger, \\
(E\langle G/X\rangle \mid E')^\dagger &\stackrel{\text{def}}{=} (E\langle G/X\rangle)^\dagger \mid E', \\
((a)E\langle G/X\rangle)^\dagger &\stackrel{\text{def}}{=} (a)(E\langle G/X\rangle)^\dagger, \\
(E' + E\langle G/X\rangle)^\dagger &\stackrel{\text{def}}{=} (E\langle G/X\rangle)^\dagger, \\
(E\langle G/X\rangle + E')^\dagger &\stackrel{\text{def}}{=} (E\langle G/X\rangle)^\dagger, \\
(\mu Z.E\langle G/X\rangle)^\dagger &\stackrel{\text{def}}{=} (E\langle G/X\rangle)^\dagger\{\mu Z.E\langle G/X\rangle/Z\}.
\end{aligned}$$

Intuitively $(E\langle G/X\rangle)^\dagger$ is what remains if $E\langle G/X\rangle$ performs an action induced by G. Using this notation, we may describe the operational behaviors of expressions in a more specific manner. We shall identify $E\langle X\rangle^\dagger$ to $(E\langle X/X\rangle)^\dagger$.

Lemma 3. *The following statements are valid.*
(i) If $E\langle G/X\rangle \xrightarrow{\lambda} E'$ is caused by $G \xrightarrow{\lambda} G'$, then $E' \equiv (E\langle G'/X\rangle)^\dagger$.
(ii) If $E\langle G/X\rangle \xrightarrow{\lambda} E'\langle G'/X\rangle$ is caused by an action of G, then $E'\langle G'/X\rangle \sim E'\langle \mathbf{0}/X\rangle \mid G'$.

Proof. Here $\sim$ is the structural bisimilarity, see Definition 2. The proof is a simple induction on the height of derivation. Notice that the static binding mechanism is important for the induction to go through. □

A useful fact about $E\langle G/X\rangle$ is that a communication between E and G must be through a global channel name. This is formalized in the following lemma, the proof of which is again an induction on derivation.

Lemma 4. *If $E\langle G/X\rangle \xrightarrow{\tau} E'\langle G'/X\rangle$ is caused by $E\langle X\rangle \xrightarrow{a} E'\langle X\rangle$ and $G \xrightarrow{\overline{a}} G'$, then $E\langle X\rangle^\dagger \xrightarrow{a} E'\langle X\rangle$.*

Conversely if two parts of a CCS expression can perform complementary actions at a global channel, then they can interact through that channel. This property can be described by the following two lemmas.

Lemma 5 (Confluence)**.** *If $E\langle G/X\rangle \xrightarrow{\lambda_1} (E\langle G'/X\rangle)^\dagger \xrightarrow{\lambda_2} E'\langle G'/X\rangle$ is caused by a λ_1 of G and a λ_2 of $E\langle X\rangle^\dagger$, then $E\langle G/X\rangle \xrightarrow{\lambda_2}\xrightarrow{\lambda_1} E'\langle G'/X\rangle$.*

Lemma 6. *If $E\langle G/X\rangle \xrightarrow{\overline{a}} (E\langle G'/X\rangle)^\dagger \xrightarrow{a} E'\langle G'/X\rangle$ is caused by the action $G \xrightarrow{\overline{a}} G'$ and $E\langle X\rangle^\dagger \xrightarrow{a} E'$, then $E\langle G/X\rangle \xrightarrow{\tau} E'\langle G'/X\rangle$.*

2.2 Bisimulation Theory

A basic technique in the observational theory of interaction is the bisimulation technique [50, 39, 57]. The co-inductive nature of bisimulation can be seen from the following simple argument: To conclude if the two processes $E \stackrel{\rm def}{=} a.E'$ and $F \stackrel{\rm def}{=} a.F'$ are observationally equivalent, one needs to know if E' and F' are observationally equivalent. Now E', respectively F', may well be E, respectively F. It has been demonstrated that co-induction is almost a universal tool in analyzing the observational behaviors of interactions.

This subsection lays down the basic bisimulation theory of CCS. We shall focus on the CCS defined above. However the results in this subsection are valid for all the variants of CCS considered in this paper.

Definition 1. *A binary symmetric relation $\mathcal{R}$ on $\mathcal{E}_{\rm CCS}$ is a* bisimulation *if the following statements are valid whenever $E\mathcal{R}F$.*
(i) $E\varsigma\mathcal{R}F\varsigma$ for every substitution ς of process variables.
(ii) If $E \xrightarrow{\lambda} E'$ then $F \xRightarrow{\widehat{\lambda}} F'\mathcal{R}E'$ for some F'.
The bisimilarity $\approx$ is the largest bisimulation.

The stronger version of $\approx$ is often useful.

Definition 2. *A binary symmetric relation $\mathcal{R}$ on $\mathcal{E}_{\rm CCS}$ is a* structural bisimulation *if the following statements are valid whenever $E\mathcal{R}F$.*
(i) $E\varsigma\mathcal{R}F\varsigma$ for every substitution ς of process variables.
(ii) If $E \xrightarrow{\lambda} E'$ then $F \xrightarrow{\lambda} F'\mathcal{R}E'$ for some F'.
The structural bisimilarity $\sim$ is the largest structural bisimulation.

The equivalence relations $\sim$ and $\approx$ are defined on the set of the expressions with free variables. It follows immediately from Definition 2 and Definition 1 that both $\sim$ and $\approx$ are closed under the substitutions on variables. Moreover the two relations are also closed under injective renaming.

Lemma 7. *If $E \approx F$ then $E\alpha \approx F\alpha$ for every injective renaming α.*

If $\mathcal{R}$ is a symmetric relation on CCS processes, the condition (i) of Definition 1 and Definition 2 is vacuously satisfied. Fortunately we only have to consider relations of this kind when establishing bisimulation equality for process expressions. This is due to the following lemma.

Lemma 8. *$E \approx F$ if and only if $E\{a/X\} \approx F\{a/X\}$ for some name a that does not appear in $E \,|\, F$.*

Proof. One direction is obvious. For the other direction let $\mathcal{S}$ be the following relation

$$\left\{ (E\{G/X\}, F\{G/X\}) \;\middle|\; \begin{array}{l} E\{a/X\} \approx F\{a/X\}, \\ G \text{ is a process expression,} \\ a \text{ does not appear in } E \,|\, F \,|\, G \end{array} \right\}.$$

It follows from Lemma 7 that $\mathcal{S}$ is closed under substitution of variables. Now suppose $E\{G/X\} \xrightarrow{\lambda} H$. There are three cases.

- The action $E\{G/X\} \xrightarrow{\lambda} E'\{G/X\}$ is caused by $E \xrightarrow{\lambda} E'$. Obviously $E\{a/X\} \xrightarrow{\lambda} E'\{a/X\}$ must be simulated by $F\{a/X\} \stackrel{\widehat{\lambda}}{\Longrightarrow} F'\{a/X\} \approx E'\{a/X\}$ for some F'. Hence $F\{G/X\} \stackrel{\widehat{\lambda}}{\Longrightarrow} F'\{G/X\}\,\mathcal{S}^{-1}\,E'\{G/X\}$.

- The action $E\langle G\rangle\{G/X\} \xrightarrow{\lambda} (E\langle G'/X\rangle)^{\dagger}\{G/X\}$ is caused by $G \xrightarrow{\lambda} G'$. Then $E\langle a/X\rangle\{a/X\} \xrightarrow{a} (E\langle \mathbf{0}/X\rangle)^{\dagger}\{a/X\}$, which must be simulated by

$$F\langle a/X\rangle\{a/X\} \stackrel{a}{\Longrightarrow} F'\langle \mathbf{0}\rangle\{a/X\} \approx E'\langle \mathbf{0}/X\rangle\{a/X\}$$

for some F'. Now

$$\begin{array}{rcl} F\langle G/X\rangle\{a/X\} & \stackrel{\lambda}{\Longrightarrow} & F'\langle G'/X\rangle\{a/X\} \\ & \sim & F'\langle \mathbf{0}/X\rangle\{a/X\}\,|\,G' \\ & \approx & E'\langle \mathbf{0}/X\rangle\{a/X\}\,|\,G' \\ & \sim & E'\langle G'/X\rangle\{a/X\} \end{array}$$

where the structural bisimilarities are due to Lemma 3. Therefore

$$F\langle G/X\rangle\{G/X\} \stackrel{\lambda}{\Longrightarrow} F'\langle G'/X\rangle\{G/X\}\,\mathcal{S}^{-1}\,E'\langle G'/X\rangle\{G/X\}.$$

- The action $E\langle G/X\rangle\{G/X\} \xrightarrow{\tau} E'\langle G'/X\rangle\{G/X\}$ is caused by $E\langle X\rangle^{\dagger} \xrightarrow{b} E'\langle X\rangle$ and $G \xrightarrow{\overline{b}} G'$. Therefore

$$E\langle a/X\rangle\{a/X\} \xrightarrow{a} (E\langle \mathbf{0}/X\rangle)^{\dagger}\{a/X\} \xrightarrow{b} E'\langle \mathbf{0}/X\rangle\{a/X\}$$

which must be simulated by $F\langle a/X\rangle\{a/X\}$ as follows:

$$\begin{array}{rcl} F\langle a/X\rangle\{a/X\} & \Longrightarrow & F_1\langle a/X\rangle\{a/X\} \\ & \xrightarrow{a} & F_2\langle \mathbf{0}/X\rangle\{a/X\} \\ & \Longrightarrow & F_3\langle \mathbf{0}/X\rangle\{a/X\} \\ & \xrightarrow{b} & F_4\langle \mathbf{0}/X\rangle\{a/X\} \\ & \Longrightarrow & F_5\langle \mathbf{0}/X\rangle\{a/X\} \\ & \approx & E'\langle \mathbf{0}/X\rangle\{a/X\}. \end{array}$$

It follows from Lemma 5 that some F_2' exists such that

$$\begin{array}{rcl} F_1\langle a/X\rangle\{a/X\} & \Longrightarrow & F_2'\langle a/X\rangle\{a/X\} \\ & \xrightarrow{a} & F_3\langle \mathbf{0}/X\rangle\{a/X\}. \end{array}$$

Lemma 6 and the following two-action sequence

$$\begin{array}{rcl} F_2'\langle G/X\rangle\{a/X\} & \xrightarrow{\overline{b}} & F_3\langle G'/X\rangle\{a/X\} \\ & \xrightarrow{b} & F_4\langle G'/X\rangle\{a/X\} \end{array}$$

imply that $F_2'\langle G/X\rangle\{a/X\} \xrightarrow{\tau} F_4\langle G'/X\rangle\{a/X\}$. Hence

$$\begin{array}{rcl} F\langle G/X\rangle\{a/X\} & \Longrightarrow & F_5\langle G'/X\rangle\{a/X\} \\ & \sim & F_5\langle \mathbf{0}/X\rangle\{a/X\} \,|\, G' \\ & \approx & E'\langle \mathbf{0}/X\rangle\{a/X\} \,|\, G' \\ & \sim & E'\langle G'/X\rangle\{a/X\}. \end{array}$$

Consequently

$$\begin{array}{rcl} F\langle G/X\rangle\{G/X\} & \Longrightarrow & F_5\langle G'/X\rangle\{G/X\} \\ & \mathcal{S}^{-1} & E'\langle G'/X\rangle\{G/X\}. \end{array}$$

We conclude that $\mathcal{S}$ is a bisimulation. □

Lemma 8 allows one to concentrate on the bisimulations on processes. It also indicates that our bisimilarity is the same as Milner's bisimilarity on CCS expressions [39].

Definition 3. *A binary symmetric relation $\mathcal{R}$ on $\mathcal{E}_{\mathrm{CCS}}$ is a* bisimulation up to $\sim$ *if the following statements are valid whenever $E\mathcal{R}F$.*
(i) $E\varsigma\mathcal{R}F\varsigma$ for every substitution ς of process variables.
(ii) If $E \xrightarrow{\lambda} E'$ then $F \overset{\widehat{\lambda}}{\Longrightarrow} F' \sim \mathcal{R} \sim E'$ for some F'.

The reason to introduce the above definition is the following useful fact.

Lemma 9. *If $\mathcal{R}$ is a bisimulation up to $\sim$ then $\mathcal{R} \subseteq \approx$.*

Before ending this subsection we state a result that will be referred to later on. See [39] for detail.

Lemma 10. *If every occurrence of X in E is bounded by some prefix operation, then the equation $X \sim E$ has a unique solution up to $\sim$.*

2.2.1 Subbisimilarity

To study the expressive power of CCS variants, we would like to introduce a tool that embodies both Operational Correspondence and Observational Correspondence. The *Equivalence Criterion* has been frequently used to compare the relative expressive powers of the variants of a same model, see for example [49]. Given an equivalence relation $\asymp$, Equivalence Criterion can be stated as follows:

> $\mathcal{L}_2$ is at least as expressive as $\mathcal{L}_1$ if for every P_1 in $\mathcal{L}_1$ there is some P_2 in $\mathcal{L}_2$ such that $P_1 \asymp P_2$.

To apply this criterion, one needs to pin down an equivalence relation. It is our view that the bisimilarity of Park [50] and Milner [39] offers the right balance between the distinguishing power of the observer and the control power of the observee. So most of the time we shall understand $\asymp$ as the weak bisimilarity

$\approx$. To prove that a process P_1 of $\mathcal{L}_1$ is bisimilar to a process P_2 of $\mathcal{L}_2$, one actually constructs a binary relation that contains the pair (P_1, P_2) such that the bisimulation property holds of the relation. We shall call such a relation a subbisimilarity. Let's start with the following simple definition.

Definition 4. *A relation $\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is* total *if $\forall S_1 \in \mathcal{S}_1.\exists S_2 \in \mathcal{S}_2.S_1\mathcal{R}S_2$. It is* surjective *if $\forall S_2 \in \mathcal{S}_2.\exists S_1 \in \mathcal{S}_1.S_1\mathcal{R}S_2$.*

We will write $\mathcal{R};\mathcal{R}'$ for the composition of $\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ and $\mathcal{R}' \subseteq \mathcal{S}_2 \times \mathcal{S}_3$, and $\mathcal{R}^{-1}$ for the reverse relation of $\mathcal{R}$.

Intuitively a subbisimilarity interprets every process of a variant by (at least) one process of another variant in such a way that generalizes an encoding. The following definition strengthens the definition of bisimulation of [27] with the totality condition.

Definition 5. *Suppose* CCS^1 *and* CCS^2 *are two* CCS *variants. A* subbisimilarity *from* CCS^1 *to* CCS^2 *is a total relation $\mathcal{R} \subseteq \mathcal{P}_{\mathrm{CCS}^1} \times \mathcal{P}_{\mathrm{CCS}^2}$ such that the following* bisimulation property *holds whenever $P_1\mathcal{R}P_2$.*

If $P_1 \xrightarrow{\lambda}_1 P_1'$ then $P_2 \overset{\widehat{\lambda}}{\Longrightarrow}_2 P_2'\mathcal{R}^{-1}P_1'$ for some P_2'.

If $P_2 \xrightarrow{\lambda}_2 P_2'$ then $P_1 \overset{\widehat{\lambda}}{\Longrightarrow}_1 P_1'\mathcal{R}P_2'$ for some P_1'.

We shall say that CCS^1 *is* subbisimilar *to* CCS^2*, notation* CCS^1 $\sqsubseteq^{ccs}$ CCS^2*, if there is a subbisimilarity from* CCS^1 *to* CCS^2*. We write* CCS^1 $\sqsubset^{ccs}$ CCS^2 *if* CCS^1 $\sqsubseteq^{ccs}$ CCS^2 *and* CCS^2 $\not\sqsubseteq^{ccs}$ CCS^1.

It is obvious that the subbisimilarity relationship $\sqsubseteq^{ccs}$ is transitive. In what follows, we often say that an interpretation $[\![\_]\!]$ from CCS^1 to CCS^2 is a subbisimilarity. This should be understood as saying that $[\![\_]\!]$ generates a subbisimilarity. To prove CCS^2 $\not\sqsubseteq^{ccs}$ CCS^1, one only has to show that some P_2 of CCS^2 exists such that for every P_1 of CCS^1 the pair P_1, P_2 do not satisfy the bisimulation property. Equivalently it amounts to showing the following property: $\exists P_2 \in \mathcal{P}_{\mathrm{CCS}^2}.\forall P_1 \in \mathcal{P}_{\mathrm{CCS}^1}.P_2 \not\approx P_1$.

Our particular choice of subbisimilarity can be further justified in the following accounts:

- The subbisimilarities are based on the assumption that the environments are dynamically changing and therefore are strong enough to detect the difference of two processes that are not bisimilar. This is a reasonable assumption for a model of interaction.
- A positive result stated in terms of subbisimilarity, say CCS^1 $\sqsubseteq^{ccs}$ CCS^2, is more significant than most positive statements using other weaker criteria.
- A counter example refuting the existence of a subbisimilarity can often be modified to give a separation result when a set of weaker criteria is applied. The negative results stated in terms of subbisimilarity are far more general than they appear to be.

Our motivation is that the subbisimilarities should play a role similar to that of the reductions in computing theory. This point is elaborated next.

2.2.2 Subbisimilarity and Church-Turing Thesis

The theory of computation is built upon a fundamental assumption (hypothesis) referred to as Church-Turing Thesis [35, 16]. It is normally stated as follows.

> *Church-Turing Thesis*: Two models of computation, $\mathcal{M}_1$ and $\mathcal{M}_2$, are equivalent in the sense that the functions definable in $\mathcal{M}_1$ are definable in $\mathcal{M}_2$ and vice versa.

Basically Church-Turing Thesis says that the theory of computation is model-independent. Various extensions of the thesis have been proposed with the development of the theory of algorithm [2] and the complexity theory [64]. The Extended Church-Turing Thesis is a generalization of Church-Turing Thesis. It states that two models of computation not only define the same set of functions, they also simulate each other's calculations of functions. The following formulation is taken from [64].

> *Extended Church-Turing Thesis*: For two models of computation, $\mathcal{M}_1$ and $\mathcal{M}_2$, there is a polynomial p such that t computation steps of $\mathcal{M}_1$ on an input of length n can be simulated by $p(n,t)$ computation steps of $\mathcal{M}_2$.

The Extended Church-Turing Thesis ensures that the bulk of the complexity theory is model-independent. The practical implication of this thesis is that a compiler never changes the feasibility of programs. When moving from compilers to interpreters, another aspect of Church-Turing Thesis comes in sight. An interpreter should not introduce too much computational overhead to the extent that a feasible solution is executed in an infeasible way. This is a strong version of Church-Turing Thesis.

> *Strong Church-Turing Thesis*: For two models of computation, $\mathcal{M}_1$ and $\mathcal{M}_2$, there is a polynomial p such that a program of $\mathcal{M}_1$ of size n is translated mechanically to a program of $\mathcal{M}_2$ in $p(n)$ steps and that the program and its translation simulate each other's computations.

Let's try to formalize the computational and the effective aspects of Church-Turing Thesis in an abstract setting. Suppose that $(\mathcal{M}_0, \rightarrow_0)$ and $(\mathcal{M}_1, \rightarrow_1)$ are two models of computation. Let $\rightarrow_i^*$, for $i \in \{0,1\}$, be the reflexive and transitive closure of $\rightarrow_i$. If we ignore the feasibility requirement, Church-Turing Thesis means at least that there is a recursive function that translates a program M of $\mathcal{M}_0$ to a program N of $\mathcal{M}_1$. A computational step $M \rightarrow_0 M'$ is simulated by several computation steps $N \rightarrow_1^* N'$. It often happens that the program N' is syntactically different from the translation of M'. But the two are equivalent semantically. A standard approach to get around this problem is to introduce some equivalence relation on the set of all the programs of $\mathcal{M}_1$. If $\mathcal{M}_1$ say is the λ-calculus, the β conversion $=_\beta$ is often taken to be the equivalence. In some other cases a structural congruence is introduced to play the role. The use of the equivalence relation is orthogonal to the interpretation. There is not any canonicity in the choice of such an equivalence relation. A better approach is to extend the interpretation from a recursive function to a relation that inherits the effectiveness of the interpretation function and reincarnates the equivalence on the programs of $\mathcal{M}_1$. So what Church-Turing Thesis tells us is that there should be a total relation $\mathfrak{I}$ from the set of the programs of $\mathcal{M}_0$ to the set of the programs of $\mathcal{M}_1$ such that given a program M of $\mathcal{M}_0$ the set of the interpretations of M under $\mathfrak{I}$ can be effectively generated. In both programming practice and in theory, another aspect of effectiveness is necessary. Suppose $\mathfrak{I}$ is an interpreter and $M_0\mathfrak{I}M_1$. If M_1 evaluates to M_1' after a finite number of computations steps, then the interpreter needs to find effectively some M_0' such that $M_0'\mathfrak{I}M_1'$. After finding out M_0' the interpreter may return M_0' as the result of the evaluation of M_0. A mild formalization of these effectiveness requirements is given in the next definition.

Definition 6. *A binary relation $\mathcal{R}$ from an effective set $\mathcal{S}$ to an effective set $\mathcal{T}$ is* effective *if the following statements are valid: (i) If $\mathcal{T}'$ is an effective subset of $\mathcal{T}$, then $\{x \mid (x,y) \in \mathcal{R} \wedge y \in \mathcal{T}'\}$ is effective. (ii) If $\mathcal{S}'$ is an effective subset of $\mathcal{S}$, then $\{y \mid (x,y) \in \mathcal{R} \wedge x \in \mathcal{S}'\}$ is effective.*

To appreciate Definition 6, recall that a partial numeric function $f : X \rightharpoonup Y$ is effective if and only if its graph is effective. If f is effective and X' is an effective subset of X then $f(X')$ is effective. Conversely if f is effective and Y' is an effective subset of Y then $f^{-1}(Y')$ is effective. The effective relations are meant to generalize the effective functions.

Let $\mathfrak{I}$ be a total relation from $\mathcal{P}_{\mathcal{M}_0}$ to $\mathcal{P}_{\mathcal{M}_1}$ that demonstrates the Turing completeness of $\mathcal{M}_1$ assuming that $\mathcal{M}_0$ is Turing complete. The relation $\mathfrak{I}$ should validate at least the following statements.

1. If $M\mathfrak{I}N$ and $M \rightarrow_0 M'$ then $N \rightarrow_1^* N'\mathfrak{I}^{-1}M'$ for some N'.
2. If $M\mathfrak{I}N \rightarrow_1 N'$ then $M \rightarrow_0^* M'\mathfrak{I}N'$ for some M'.
3. If $M_0\mathfrak{I}N_0$ and $M_0 \rightarrow_0 M_1 \rightarrow_0 \ldots M_i \rightarrow_0 \ldots$ is an infinite computation sequence, then there must be some $k \geq 1$ and N' such that $N_0 \rightarrow_1^+ N'\mathfrak{I}^{-1}M_k$.
4. If $M_0\mathfrak{I}N_0$ and $N_0 \rightarrow_1 N_1 \rightarrow_1 \ldots N_i \rightarrow_1 \ldots$ is an infinite computation sequence, then there must be some $k \geq 1$ and M' such that $M_0 \rightarrow_0^+ M'\mathfrak{I}N_k$.
5. *Effectiveness Condition*: $\mathfrak{I}$ is effective.

In the statement of Effectiveness Condition, we have confused for example the set $\mathcal{P}_{\mathcal{M}_0}$ with the set of the Gödel numberings of the elements of $\mathcal{P}_{\mathcal{M}_0}$. No harm should be caused by this confusion.

Conditions 1 and 2 follow from Extended Church-Turing Thesis. It is important to realize that the underlying philosophy of Extended Church-Turing Thesis is that equivalence of computations is of a bisimulation nature. Conditions 3 and 4 are required by Church-Turing Thesis. This is because a program that ends with a result upon an input is different from any program that fails to deliver anything after receiving the same input. To go along with Strong Church-Turing Thesis, Conditions 3 and 4 are formulated in a bisimulation style.

Condition 5 reminds one of Turing reduction. Without Effectiveness Condition the requirements 1-4 are too weak to be of any use. Consider the following translation from the Deterministic Turing Machines to the λ-terms. It maps all the terminating machines to $\mathbf{I} \stackrel{\mathrm{def}}{=} \lambda x.x$ and all the nonterminating machines to $\mathbf{\Omega} \stackrel{\mathrm{def}}{=} (\lambda x.xx)(\lambda x.xx)$. This map satisfies the conditions 1-4 stated above. However in no sense can it be used to demonstrate the Turing completeness of the λ-calculus.

Now how are the computations modeled in theory of interaction? The guideline is that, since computations are internal actions, they are modeled by the τ-actions. If a process interprets a program, then all the internal actions of the process correspond to the computations of the program. From the observational point of view, since computations are calculations or evaluations, if $M \rightarrow^* M'$ then the interpretations $\widehat{M}, \widehat{M'}$ of M, M' respectively in a process calculus should be equivalent interactively. In other words, we have the following situation:

$$\widehat{M} \xrightarrow{\tau} \ldots \xrightarrow{\tau} \widehat{M'} \approx \widehat{M}.$$

In theory of interaction, what are the rules that govern the behaviors of the above computation? If M is calculated half way through to an intermediate state M'', our intuition about Turing computation suggests that $\widehat{M''}$ must be equivalent to $\widehat{M}$. In theory of interaction this is guaranteed [19].

Lemma 11 (Computation Lemma). *If $P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_n \approx P_0$ then $P_0 \approx P_1 \approx P_2 \approx \cdots \approx P_n$.*

Although Computation Lemma is stated for $\approx$, it holds for almost all the observational equivalences one can think of. Computation Lemma is a property about the internal actions of systems, it is not a property about process equivalences.

According to Computation Lemma, the self evolution of a system is carried out in phases, which can be depicted as follows.

$$P_0 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_{i_1} \xrightarrow{\tau} P_{i_1+1} \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_{i_2} \xrightarrow{\tau} \ldots.$$

The phase one consists of all the states from P_0 to P_{i_1}, which are all equivalent from the point of view of interaction. What phase one accomplishes is some internal adjustment. It has not made any irretrievable decision. The step from P_{i_1} to P_{i_1+1} is fundamental. Once the system has done that, there is no going back. This pattern is repeated again and again throughout the evolution.

Computation Lemma is of crucial importance to the encodings studied in this paper. A typical scenario of applying Computation Lemma is the following:

$$\begin{tikzcd}
{} & & P_0 \arrow[r, "\lambda_0"] \arrow[dr, bend left, "\tau"] & {} & \\
& P_n \arrow[ul, "\lambda_n"] \arrow[ur, bend left, "\tau"] & & P_1 \arrow[dr, "\lambda_1"] \arrow[dl, dashed, bend left, "\tau^*"] & \\
{} & & P_i \arrow[ll, "\lambda_i"'] \arrow[ul, dashed, bend left, "\tau^*"] & & {}
\end{tikzcd}$$

Imagine that there is an agent traveling around the states of the circle. All the states on the circle are equivalent by Computation Lemma. So as long as the agent is staying on the circle, it retains all the powers to kick off any of the possible actions doable by P_i. But at any particular state, say P_i, the agent might decide to kick off the action λ_i. Once it does that, the system moves to the next stage. The circle is a reiterative infinite computation. It will be proved that for many encodings this kind of infinite computation has to be introduced.

While on the subject, we mention that the fact stated in Computation Lemma has been explored in the definition of branching bisimilarity [26] of van Glabbeek and Weijland. It is argued in [26] that the branching bisimilarity, which is stronger than $\approx$, should be preferred to $\approx$ in a number of applications. As a matter of fact encoding between process calculi is an important application where the branching bisimilarity has an edge over the bisimilarity of Park and Milner.

Once Church-Turing Thesis has been formalized in the above manner, the close relationship between Church-Turing Thesis and subbisimilarity emerges. Now suppose $\mathcal{R}$ is a subbisimilarity from CCS^1 to CCS^2. Let $\mathcal{R}^c$ be the least superset of $\mathcal{R}$ that satisfies the following closure properties:

1. If $P_1 \mathcal{S} P_2$ then $(c)P_1 \mathcal{S} (c)P_2$;

2. If $P_1 \mathcal{S} P_2$ and P is in both CCS^1 and CCS^2, then $P_1 \,|\, P \;\mathcal{S}\; P_2 \,|\, P$ and $P \,|\, P_1 \;\mathcal{S}\; P \,|\, P_2$.

These closure properties are elementary from the point of view of interaction. So $\mathcal{R}^c$ must satisfy the conditions 1 and 2 given after Definition 6. It should also meet the following primary condition of observation:

3. If $P_1 \mathcal{S} P_2$ then P_1 can do a non-τ-action if and only if P_2 can do a non-τ-action.

Using a routine argument, it is easy to show that if $\mathcal{R}$ satisfies the conditions 1, 2 and 3 just stated, then the composition $\sim;\mathcal{R};\sim$ is a subbisimilarity.

The conclusion is that, from the viewpoint of Church-Turing Thesis, the subbisimilarities are generalizations from the interpretations between the computation models to the interpretations between the interaction models in a right direction. The generalizations are not completely faithful in that both divergence and effectiveness are neglected. This is why we need to strengthen the definition of subbisimilarity later.

2.3 Choice

In practice the general choice operator $+$ is rarely used. It is often sufficient to use the *mixed choice*

$$\sum_{i\in I}\lambda_i.E_i$$

where the indexing set I is finite. From a programming point of view, the *separated choice*

$$\sum_{i\in I}a_i.E_i \text{ or } \sum_{i\in I}\overline{a}_i.E_i$$

is even better. The operational semantics of the mixed (separated) choice is defined by the following rule.

$$\frac{}{\sum_{i\in I}\lambda_i.E_i \xrightarrow{\lambda_i} E_i}$$

It is worth remarking that $a.E+\tau.F$ for example can be defined in terms of the separated choices by $(b)((a.E+b.F)\,|\,\overline{b})$, where b is fresh.

We write CCS^m and CCS^s for the CCS with the mixed choice, respectively the separated choice. In both CCS^m and CCS^s the prefix operator could be dropped. The power of choice can be further restrained by removing all forms of choice operator altogether. Let CCS$^-$ be the CCS without the choice operator. The actions of CCS$^-$ satisfy the following diamond property.

Lemma 12. *Suppose E is in* CCS$^-$*. If $E \xrightarrow{\lambda} E'$ and $E \xrightarrow{\lambda'} E''$ such that $\lambda \neq \lambda'$, λ is not an interaction at $n(\lambda')$ and λ' is not an interaction at $n(\lambda)$, then $E' \xrightarrow{\lambda'} E'''$ and $E'' \xrightarrow{\lambda} E'''$ for some E'''.*

Proof. This is a simple induction on derivations. □

Using similar induction one could prove similar result for CCS^s.

Lemma 13. *Suppose E is a* CCS^s *expression. If $E \xrightarrow{a} E'$ and $E \xrightarrow{\overline{b}} E''$, then $E' \xrightarrow{\overline{b}} E'''$ and $E'' \xrightarrow{a} E'''$ for some E'''.*

It comes as no surprise that CCS$^-$ is strictly less expressive than CCS^s in terms of interactability. What is surprising is that CCS^s and CCS^m are equivalent interactively.

Proposition 1. CCS$^-$ $\sqsubset^{ccs}$ CCS^m $\sqsubseteq^{ccs}$ CCS^s.

Proof. It is clear that CCS$^-$ $\sqsubseteq^{ccs}$ CCS^m. We have to show that CCS^m $\not\sqsubseteq^{ccs}$ CCS$^-$. Suppose that P is a process containing no choice operations and that $a{+}b \approx P$. To match up the action $a{+}b \xrightarrow{a} \mathbf{0}$ there must be some P' such that $P \Longrightarrow P' \xrightarrow{a}$. Now $P' \approx a{+}b$. Therefore P'' exists such that $P' \Longrightarrow P'' \xrightarrow{b}$. Since P' contains no choice operator, it follows from Lemma 12 that $P'' \xrightarrow{a}$. But then it follows from the same lemma that $P'' \xrightarrow{a}\xrightarrow{b}$, which contradicts the fact that $P'' \approx a{+}b$.

Next we show that CCS^m $\sqsubseteq^{ccs}$ CCS^s. It is enough to explain how the encoding of the mixed choice is defined. Let R be $a.P{+}\overline{b}.Q$. The interpretation of R is defined by the following induction.

$$[\![R]\!] \quad \overset{\text{def}}{=} \quad a.P{+}\tau.\mu Y.(\overline{b}.Q{+}\tau.(a.P{+}\tau.Y)). \tag{1}$$

Intuitively $[\![R]\!]$ and $\mu Y.(\overline{b}.Q{+}\tau.(a.P{+}\tau.Y))$ satisfy the following equations

$$\begin{array}{rcl} X & \approx & a.P{+}\tau.Y, \\ Y & \approx & \overline{b}.Q{+}\tau.X. \end{array}$$

Obviously $[\![R]\!] \approx a.P{+}\overline{b}.Q$ in CCS^m and $[\![R]\!]$ is in CCS^s. This is a simple application of Computation Lemma. □

It is well known that CCS with guarded recursion is finite branching. A slightly weaker restriction, the mixed choice, however is not sufficient to guarantee the finite branching property. The infinite branching property is important when defining a process that behaves like a *random timer*. When left alone, a random timer ticks on its own. If a process sets the timer by interacting with it at a particular interface, the timer terminates after a finite number of ticks. The number of ticks is set at random. In CCS a random timer can be defined as follows:

$$RanTimer \overset{\text{def}}{=} \mu X.(s + tick \,|\, X). \tag{2}$$

RanTimer can be set at the interface s. To see how it works, notice that the following derivation is admissible.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{s{+}tick \,|\, \mu X.(s{+}tick \,|\, X) \xrightarrow{s} \mathbf{0}}{\mu X.(s{+}tick \,|\, X) \xrightarrow{s} \mathbf{0}}}{tick \,|\, \mu X.(s{+}tick \,|\, X) \xrightarrow{s} tick \,|\, \mathbf{0}}}{s{+}tick \,|\, \mu X.(s{+}tick \,|\, X) \xrightarrow{s} tick \,|\, \mathbf{0}}}{\mu X.(s{+}tick \,|\, X) \xrightarrow{s} tick \,|\, \mathbf{0}}}{\vdots}}{\mu X.(s{+}tick \,|\, X) \xrightarrow{s} tick \,|\, \ldots \,|\, tick \,|\, \mathbf{0}}$$

Another way to look at the operational behavior of *RanTimer* is to unfold it for i times. It is the following process

$$\underbrace{s{+}tick \,|\, (s{+}tick \,|\, (\ldots \,|\, (s}_{i \text{ times}}{+}tick \,|\, \mu X.(s{+}tick \,|\, X))\ldots)).$$

So an action sequence of *RanTimer* typically looks like this:

$$RanTimer\ \underbrace{\xrightarrow{tick}\ldots\xrightarrow{tick}}_{i \text{ times}}\xrightarrow{s}\underbrace{\xrightarrow{tick}\ldots\xrightarrow{tick}}_{j \text{ times}}\mathbf{0}.$$

After *RanTimer* has been set at s, it may turn into one of an infinite number of semantically different processes. This property makes the random timer a prime counter example to separate the expressive powers of process calculi. The random timer is about ability to make a choice for the future, when the potential choices are infinite.

Lemma 14. CCS^m*,* CCS^s *and* CCS *are infinite branching.*

Proof. RanTimer, defined in CCS, is infinite branching. The following transition

$$\mu X.(a\,|\,X)\xrightarrow{a}a\,|\,\ldots\,|\,a\,|\,\mu X.(a\,|\,X)$$

shows that both CCS^m and CCS^s are infinite branching as well. □

As long as the recursion is not guarded, infinite branching is present. But there is a crucial difference between the two examples given in the proof of Lemma 14. All the derivatives of *RanTimer* after performing s are semantically different from one to another. On the other hand all the derivatives of $\mu X.(a\,|\,X)$ are semantically equivalent. The latter fact is actually a particular instance of a general phenomenon. To establish the fact, we need some technical lemmas.

Lemma 15. *Suppose $E\langle X\rangle, F\langle X\rangle$ are expressions in* CCS^m *and that both the specified occurrence of X in $E\langle X\rangle$ and the specified occurrence of X in $F\langle X\rangle$ are unguarded. Then $E\langle F\langle X\rangle\rangle \sim F\langle E\langle X\rangle\rangle$.*

Proof. Suppose that E is in CCS^m and that $E\langle X\rangle$ indicates an unguarded occurrence of X. By structural induction it is easy to show that $E\langle X\rangle \sim E\langle\mathbf{0}\rangle\,|\,X$. Hence

$$\begin{aligned}E\langle F\langle X\rangle\rangle &\sim E\langle\mathbf{0}\rangle\,|\,F\langle X\rangle\\ &\sim E\langle\mathbf{0}\rangle\,|\,F\langle\mathbf{0}\rangle\,|\,X\\ &\sim F\langle\mathbf{0}\rangle\,|\,E\langle X\rangle\\ &\sim F\langle E\langle X\rangle\rangle.\end{aligned}$$

We are done. □

If an action of $(\mu Z.G)\{E/X\}$ is caused by a copy of E, then the action must be caused by an occurrence of E in the unfolding of $(\mu Z.G)\{E/X\}$. This intuition is formalized by the next lemma.

Lemma 16. *Suppose $G[X], E, F$ are expressions in* CCS^m*. If $G[E]\xrightarrow{\lambda}F$ is caused by an occurrence of E, then there exist some $G_1\langle X\rangle, E'$, where the specified X is unguarded, such that $G\sim G_1\langle X\rangle$ and $G_1\{E/X\}\langle E/X\rangle\xrightarrow{\lambda}G_1\{E/X\}\langle E'/X\rangle\equiv F$ is caused by $E\xrightarrow{\lambda}E'$.*

Proof. Suppose $G[E] \xrightarrow{\lambda} F$ is caused by an occurrence of E. We may prove the lemma by induction on derivation.

- $G \equiv X$. This case is obvious.
- G cannot be a choice since all the choices are guarded in CCS^m.
- $G \equiv (a)G'$ or $G \equiv G' \,|\, G''$. These cases are simple by induction.
- $G \equiv \mu Z.G'$. Then $G'\{\mu Z.G'/Z\}[E] \xrightarrow{\lambda} F$ has a shorter derivation. By induction, there exist some $G_1\langle X\rangle, E'$ such that the specified occurrence of X is unguarded, $G'\{\mu Z.G'/Z\} \sim G_1\langle X\rangle$ and $G_1\{E/X\}\langle E/X\rangle \xrightarrow{\lambda} G_1\{E/X\}\langle E'/X\rangle \equiv F$ is caused by $E \xrightarrow{\lambda} E'$. But then $G \sim G_1\langle X\rangle$.

We are done. □

An action of $E[E]$ is caused either by $E[X]$ or by an occurrence of E that substitutes for X. What is interesting is that the latter situation can be avoided in favor of the former.

Lemma 17. *The following statements are valid for every $E[X]$ of CCS^m:*
(i) If $E \xrightarrow{\lambda} F$ then $E[F[\mu X.E]] \sim F[\mu X.E]$.
(ii) If $E[E] \xrightarrow{\lambda} F$ then $E[F[\mu X.E]] \sim F[\mu X.E]$.

Proof. We focus on the proof of (ii). There are several cases since the action could be caused by an occurrence of E or by an interaction. Let's take a look at two major cases.

- The action $E[E] \xrightarrow{\lambda} F$ is caused by an occurrence of the outer E. By Lemma 16, there exist some $E_1\langle X\rangle, E'$, where the specified occurrence of X is unguarded, $E \sim E_1\langle X\rangle$ and

  $$E_1\{E/X\}\langle E/X\rangle \xrightarrow{\lambda} E_1\{E/X\}\langle E'/X\rangle \equiv F$$

  is caused by $E \xrightarrow{\lambda} E'$. It follows from $E \sim E_1\langle X\rangle$ that some $E_1'\langle X\rangle$ exists such that $E_1\langle X\rangle \xrightarrow{\lambda} E_1'\langle X\rangle \sim E'$. Now

  $$E_1\{E/X\}\langle E_1\langle X\rangle/X\rangle \xrightarrow{\lambda} E_1\{E/X\}\langle E_1'\langle X\rangle/X\rangle \sim E_1\{E/X\}\langle E'/X\rangle.$$

  Therefore

  $$\begin{array}{rcl} E[E] & \xrightarrow{\lambda} & E'[E] \\ & \sim & E_1'\{E/X\}\langle E_1\langle X\rangle/X\rangle \\ & \sim & E_1\langle E_1'\{E/X\}\langle X\rangle/X\rangle \equiv F \end{array}$$

  where the second structural bisimilarity is due to Lemma 15.

We are done. □

The above three lemmas can now be exploited to prove the following useful lemma.

Lemma 18. *Suppose $E[X]$ is in CCS^m. The following statements are valid.*
(i) If $\mu X.E \xrightarrow{\lambda} E'$ for $\lambda \neq \tau$, then $E \xrightarrow{\lambda} F$ and $F[\mu X.E] \sim E'$ for some F.
(ii) If $\mu X.E \xrightarrow{\tau} E'$ then $E[E] \xrightarrow{\tau} F$ and $F[\mu X.E] \sim E'$ for some F.

Proof. Suppose $\mu X.E \xrightarrow{\tau} E'$. By Recursion Lemma (Lemma 2),

$$\underbrace{E[E[\ldots[E}_{i \text{ times}}[X]]\ldots]] \xrightarrow{\tau} F'[X]$$

for some natural number i and some $F'[X]$ such that $F'[\mu X.E] \equiv E'$. By repeatedly using Lemma 17, we can get some $F[X]$ such that $E[E] \xrightarrow{\tau} F[X]$ and

$$F[\underbrace{E[E[\ldots[E}_{i-2 \text{ times}}[X]]\ldots]]] \sim F'[X].$$

It follows that $F[\mu X.E] \sim F'[\mu X.E] \equiv E'$. □

What Lemma 18 tells us is that the one-step derivatives of $\mu X.E$ are essentially the one-step derivatives of $E[E]$. This is the crucial property of the infinite behaviors of CCS^m. In the following proposition, $\mathfrak{Drv}^1(E)/\sim$ is the quotient of the set of the one step derivatives of E with respect to the structural bisimilarity $\sim$.

Proposition 2. *The quotient $\mathfrak{Drv}^1(E)/\sim$ is finite for every process expression E of CCS^m.*

Proof. We prove the result by a structural induction.

- If E is a mixed choice, $\mathfrak{Drv}^1(E)$ is obviously finite.
- If E is of the form $(a)E'$, a simple induction suffices.
- If E is of the form $E_1 \mid E_2$, we use induction hypothesis and the congruence property of $\sim$.
- Suppose E is of the form $\mu X.E_1$. We prove by induction on the number of the μ-operators that $\mathfrak{Drv}^1(\mu X.E_1)/\sim$ is finite. The base case is when E_1 is free of the μ-operator. By the proof of the previous cases and the assumption that E_1 does not contain any μ-operator, $\mathfrak{Drv}^1(E_1[E_1])/\sim$ is finite. It follows from Lemma 18 that $\mathfrak{Drv}^1(\mu X.E_1)/\sim$ is finite. Putting together what we have obtained so far we conclude that $\mathfrak{Drv}^1(E)/\sim$ is finite for every E containing no more than one μ-operator.

  Now assume that for all CCS^m expressions F that contain at most i μ-operators, $\mathfrak{Drv}^1(F)/\sim$ is finite. The argument is made in two steps.

- Let $F_0[X_1,..,X_m], F_1,\ldots,F_m$ be CCS^m expressions that contain at most i μ-operators. By structural induction it is easy to prove that $\mathfrak{Drv}^1(F_0[F_1/X_1,..,F_m/X_m])/\sim$ is finite.
- Suppose that E_1 contains i μ-operators. By the above property, the quotient $\mathfrak{Drv}^1(E_1[E_1])/\sim$ is finite. We conclude from Lemma 18 that $\mathfrak{Drv}^1(\mu X.E_1)/\sim$ is finite.

The structural induction is complete. □

Proposition 2 appears to be new. It is also valid of course for CCS^s.

2.3.1 Termination Preserving Encoding

The encoding (1) given in the proof of Proposition 1 is assuring from an external point of view. No environments can ever detect any difference between P and $[\![P]\!]$. There is however something lost over the translation. The process $a.P+\overline{b}.Q$ is translated to $a.P+\tau.\mu Y.(\overline{b}.Q+\tau.(a.P+\tau.Y))$ which is divergent. The point is that although the divergent computations introduced by the encoding only go through states that are equivalent to the initial state, there is some difference between a possibly divergent computation and an always terminating computation. One is interested in knowing if Proposition 1 can be strengthened to meet Computation Criterion.

The issue of divergence has been studied in the process theory. There are basically two approaches. In the explicit approach applied for example in [4, 63] a divergent process, say Ω, is introduced at the syntactical level, and a divergence predicate is defined on the structures of the processes. This predicate is then used in the definition of the equivalence relations. In the implicit approach summarized by van Glabbeek in [25] divergence is treated at the operational level. In this approach a straightforward requirement would be the following:

> If $P\mathcal{R}Q$ then P is divergent if and only if Q is divergent.

This condition completely ignores the bisimulation nature of the subbisimilarity. The right termination property that is consistent with the notion of bisimulation is the one discussed in Section 2.2.2.

Definition 7. *A subbisimilarity $\mathcal{R}$ from* CCS^1 *to* CCS^2 *is* codivergent *if it meets the Computation Criterion in the following sense.*

> *If $P\mathcal{R}Q$ and $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots$ is an infinite computation, then $Q \overset{\tau}{\Longrightarrow} Q'\mathcal{R}^{-1}P_k$ for some Q' and some $k \geq 1$.*
>
> *If $P\mathcal{R}Q$ and $Q \xrightarrow{\tau} Q_1 \xrightarrow{\tau} Q_2 \xrightarrow{\tau} \ldots$ is an infinite computation, then $P \overset{\tau}{\Longrightarrow} P'\mathcal{R}Q_k$ for some P' and some $k \geq 1$.*

We write $\mathrm{CCS}^1 \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^2$ *if a codivergent subbisimilarity from* CCS^1 *to* CCS^2 *exists, and* $\mathrm{CCS}^1 \sqsubset_{\downarrow}^{ccs} \mathrm{CCS}^2$ *if* $\mathrm{CCS}^1 \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^2$ *yet* $\mathrm{CCS}^2 \not\sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^1$.

The codivergence property defined above is essentially the eventually progressing property of Priese [52].

The next theorem says that one must be ready to sacrifice Computation Criterion if the use of the choice is restricted.

Proposition 3. $\mathrm{CCS}^s \sqsubset^{ccs}_{\downarrow} \mathrm{CCS}^m \sqsubset^{ccs}_{\downarrow} \mathrm{CCS}$.

Proof. Clearly $\mathrm{CCS}^s \sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^m \sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}$. We have to prove that there are no codivergent subbisimilarities from CCS, respectively CCS^m, to CCS^m, respectively CCS^s.

Suppose that P is a terminating process with only separated choice. We argue that P is not bisimilar to $a+\overline{b}$. If P were bisimilar to $a+\overline{b}$, then for each P' such that $P \Longrightarrow P'$ and $\neg(P' \xrightarrow{\tau})$, one would have that $P' \approx a+\overline{b}$, which would imply that $P' \xrightarrow{a}$ and $P' \xrightarrow{\overline{b}}$. This is a contradiction according to Lemma 13.

Next we prove that the random timer *RanTimer*, which is terminating, is not bisimilar to any terminating process in CCS^m. Suppose that *RanTimer* were bisimilar to some terminating P of CCS^m. Let P' be such that $P \Longrightarrow P'$ and that P' cannot do any tau action. Clearly $RanTimer \approx P'$. According to Proposition 2, there are only finitely many semantically distinct P_i, where $i \in I$ for some finite I, such that $P' \xrightarrow{s} P_i$. For each $i \in I$ there would exist some $j_i < \omega$ such that

$$RanTimer \xrightarrow{s} \underbrace{tick \,|\, \ldots \,|\, tick}_{j_i \text{ times}} \,|\, \mathbf{0} \approx P_i.$$

Let j be such that $j > \mathbf{sup}_{i\in I}\{j_i\}$. It is clear that the action $RanTimer \xrightarrow{s} \prod_j tick$ cannot be matched up by any action of P'. □

2.3.2 Mixed Choice and Guarded Recursion

In CCS^m process variables can be unguarded. One way to use a more manageable operational semantics is to use the guarded fixpoint. A fixpoint is guarded if all its *bound* process variables are guarded. Let $\mathrm{CCS}^{g\mu}$ be CCS with the guarded recursion and CCS^{gm} be CCS^m with the guarded recursion. We shall prove below that all CCS^m processes are definable in CCS^{gm}. In this subsection we will write $!E$ for $\mu Z.(E \,|\, Z)$ for some Z that does not appear in E.

The next lemma describes some of the most important properties of $!E$.

Lemma 19. $!P \sim P \,|\, !P$*;* $!!P \sim !P$*;* $!(P \,|\, Q) \sim !P \,|\, !Q$*; and* $!\mathbf{0} \sim \mathbf{0}$.

In the proof of the main result of this subsection, we shall make use of several auxiliary results about CCS^m. These results are stated in the following lemmas and corollaries.

Lemma 20. *Suppose E is a* CCS^m *expression containing X. Then* $\mu X.(X \,|\, E) \sim \mu X.(X \,|\, E\{\mathbf{0}/X\})$.

Proof. Let μ be $\mu X.(X \,|\, E)$ and μ_0 be $\mu X.(X \,|\, E\{\mathbf{0}/X\})$. Define $\mathcal{R}$ to be the relation

$$\left\{ (\prod_i \mu \,|\, G[\mu], \prod_j \mu_0 \,|\, G[\mathbf{0}]) \;\middle|\; \begin{array}{l} G[X] \text{ is in } \mathrm{CCS}^m, \\ X \text{ is guarded in } G[X], \\ \text{and } i, j \geq 1 \end{array} \right\}.$$

Now suppose that $\prod_i \mu \,|\, G[\mu] \xrightarrow{\lambda} H$. If the action is caused by $G[\mu]$, it must be caused by $G[X]$ since X is guarded in $G[X]$. Therefore some $G'[X]$ exists such that $G[X] \xrightarrow{\lambda} G'[X]$ and $\prod_i \mu \,|\, G'[\mu] \equiv H$. But then $\prod_j \mu_0 \,|\, G[\mathbf{0}] \xrightarrow{\lambda} \prod_j \mu_0 \,|\, G'[\mathbf{0}]$. If an occurrence of X in $G'[X]$ is unguarded, then by Lemma 3 one has that $G'[X] \sim G''[X] \,|\, X$ for some $G''[X]$. It follows easily from this observation that

$$\prod_i \mu \,|\, G'[\mu] \sim \mathcal{R} \sim \prod_j \mu_0 \,|\, G'[\mathbf{0}].$$

If the action $\prod_i \mu \,|\, G[\mu] \xrightarrow{\lambda} H$ is caused by a component of $\prod_i \mu$, it must be induced by some

$$\mu \xrightarrow{\lambda} \mu \,|\, E'[\mu] \tag{3}$$

which is in turn induced by $E \xrightarrow{\lambda} E'$. Now (3) can be simulated by

$$\mu_0 \xrightarrow{\lambda} \mu \,|\, E'[\mathbf{0}]. \tag{4}$$

Conversely (4) can be simulated by (3).

The above analysis suffices to show that $\mathcal{R}$ is a strong bisimulation up to $\sim$. By letting G be $\mathbf{0}$ and $i = j = 1$, we get that $\mu \sim \mu_0$. □

Lemma 20 is a powerful result. It allows one to derive a number of interesting properties of the fixpoints $\mu X.E$ whenever $E \sim X \,|\, E'$ for some E'. Some of these properties are stated in the following corollaries. The intuition behind these corollaries is that $\mu X.(X \,|\, E)$ is structurally equivalent to $!E\{\mathbf{0}/X\}$.

Corollary 1. *$\mu X.(\underbrace{X \,|\, \ldots \,|\, X}_{i \geq 1} \,|\, E) \sim \mu X.(X \,|\, E)$ if E is an expression of CCS^m.*

Corollary 2. *$\mu X.(!X \,|\, E) \sim \mu X.(X \,|\, E)$ if E is an expression of CCS^m.*

Corollary 3. *Suppose that X does not appear free in A. Then $\mu X.(X \,|\, A \,|\, E) \sim\, !A \,|\, \mu X.(X \,|\, E)$ whenever A, E are in CCS^m.*

Proof. By Lemma 20, one has that

$$\begin{array}{rcl} \mu X.(X \,|\, A \,|\, E) & \sim & \mu X.(X \,|\, A \,|\, E\{\mathbf{0}/X\}) \\ & \sim & !A \,|\, !E\{\mathbf{0}/X\} \\ & \sim & !A \,|\, \mu X.(X \,|\, E) \end{array}$$

where the second equivalence is due to Lemma 19. □

What the previous lemma and corollaries achieve is the simplification of an expression, in some particular form, underneath a fixpoint operator. If an expression E cannot be converted to some expression of the form $X \mid E'$, it is still possible to manipulate the scope of the μ-operator of $\mu X.E$. The following lemmas explain how we may go about it.

Lemma 21. *Suppose that X is guarded in E and that it does not appear free in A. Then $\mu X.(A \mid E) \sim A \mid \mu X.E\{A \mid X/X\}$.*

Proof. Let μ_1 abbreviate $\mu X.(A \mid E)$ and μ_2 abbreviate $\mu X.E\{A \mid X/X\}$. Let $\mathcal{R}$ be the following relation

$$\{(G[\mu_1], G[A \mid \mu_2]) \mid G[Z] \text{ an expression}\}\,.$$

Suppose $G[\mu_1] \xrightarrow{\lambda} H$. Since X is guarded in E, there are only three possibilities for the cause of the action.

- $G[\mu_1] \xrightarrow{\lambda} G'[\mu_1]$ is caused by $G \xrightarrow{\lambda} G'$. The simulation is $G[A \mid \mu_2] \xrightarrow{\lambda} G'[A \mid \mu_2]\ \mathcal{R}^{-1}\ G'[\mu_1]$.
- $G[\mu_1] \xrightarrow{\lambda} G'[A' \mid E[\mu_1]]$ is caused by $A \xrightarrow{\lambda} A'$. The simulation is

$$\begin{array}{rcl} G[A \mid \mu_2] & \xrightarrow{\lambda} & G'[A' \mid \mu_2] \\ & \sim & G'[A' \mid E[A \mid \mu_2]] \\ & \mathcal{R}^{-1} & G'[A' \mid E[\mu_1]]. \end{array}$$

- $G[\mu_1] \xrightarrow{\lambda} G'[A \mid E'[\mu_1]]$ is caused by $E \xrightarrow{\lambda} E'$. The simulation is

$$\begin{array}{rcl} G[A \mid \mu_2] & \xrightarrow{\lambda} & G'[A \mid E'[A \mid \mu_2]] \\ & \mathcal{R}^{-1} & G'[A \mid E'[\mu_1]]. \end{array}$$

We conclude that $\mathcal{R}$ is a structural bisimulation up to $\sim$. □

If the environment has the process expression $!R$, say a piece of universal resource, a neighboring process expression does not have to store a copy of R beside it. It can simply make use of R when necessary.

Lemma 22. *$E\{R/X\} \mid !R \sim E\{\mathbf{0}/X\} \mid !R$ whenever E is in CCS^m.*

Proof. It can be easily proved that $G \mid H\{\mathbf{0}/X\} \mid !R$ is structurally bisimilar to $G \mid H\{R/X\} \mid !R$ for all expressions G, H in CCS^m. □

In a structural approach to translations of CCS^m processes to CCS^{gm} processes, process expressions like $\mu Z.(X \mid Z)$ have to be dealt with. However the translations of such expressions have to be delayed until the variable binder μX appears. In the middle of translating a CCS^m process to a CCS^{gm} process, the process expression we get looks almost in CCS^{gm} except that it may contain subexpressions of the form $\mu Z.(X \mid Z)$. This is the reason we introduce the following terminology.

Definition 8. *A bound process variable Z in E is* quasi-guarded *if it is either guarded or is the bound process variable of an expression of the form $\mu Z.(Y \,|\, Z)$ or $\mu Z.(Z \,|\, Y)$ for some process variable Y. An expression E is quasi-guarded if all free process variables in E are guarded and all bound process variables in E are quasi-guarded. An expression is in* quasi normal form *if it is of the form*

$$(\prod_{h=1}^{k} (\underbrace{X_h \,|\, \ldots \,|\, X_h}_{i_h})) \,|\, (\prod_{g=1}^{p} !Y_g) \,|\, F$$

such that $k, p, i_1, \ldots, i_k$ are non-zero natural numbers and the following statements are valid.

1. *$X_1, \ldots, X_k, Y_1, \ldots, Y_p$ are pairwise distinct process variables.*
2. *F is quasi-guarded.*

So a quasi-guarded bound process variable is either guarded or to provide a replication of a process variable. We now prove that subexpressions like $!Y$ can be removed. For that purpose we define a syntactical operation on the process expressions by the following induction.

$$\begin{array}{rcl}
\mathbf{0}^{\bullet X} & \overset{\text{def}}{=} & \mathbf{0}, \\
Y^{\bullet X} & \overset{\text{def}}{=} & Y, \\
(E_1 \,|\, E_2)^{\bullet X} & \overset{\text{def}}{=} & E_1^{\bullet X} \,|\, E_2^{\bullet X}, \\
(\sum_{i \in I} \lambda_i.E_i)^{\bullet X} & \overset{\text{def}}{=} & \sum_{i \in I} \lambda_i.(X \,|\, E_i^{\bullet X}), \\
((a)E)^{\bullet X} & \overset{\text{def}}{=} & (a)E^{\bullet X}, \\
(\mu Z.E)^{\bullet X} & \overset{\text{def}}{=} & \mu Z.E^{\bullet X}.
\end{array}$$

The operation $(\_)^{\bullet X}$ inserts a copy of X right after every prefix. In what follows, it is only applied to process expressions in which free occurrences of X have all been removed. So we introduce the following notation:

$$E^{\circ X} \overset{\text{def}}{=} (E\{\mathbf{0}/X\})^{\bullet X}.$$

It should be clear that $E^{\circ X}$ does not contain any subexpressions of the form $!X$ and that X is guarded in $E^{\circ X}$. Operationally $E^{\bullet X}$ enjoys the property stated in the following lemma.

Lemma 23. *Suppose that E is an expression in CCS^m and that X does not appear free in E. If $E^{\bullet X} \xrightarrow{\lambda} H$ then $E \xrightarrow{\lambda} H_1$ for some H_1 such that $H \sim X \,|\, H_1^{\bullet X}$.*

Proof. Using the fact that $(E\{F/X\})^{\bullet X}$ is the same as $E^{\bullet X}\{F^{\bullet X}/X\}$, one could prove the lemma by induction on derivation. □

We can now make use of the operations $(\_)^{\bullet X}$ and $(\_)^{\circ X}$ to state some important properties about CCS^m.

Lemma 24. *If E is an expression in CCS^m and X does not appear free in E, then $!E \sim \mu X.(E^{\bullet X}) \,|\, \mu X.(E^{\bullet X})$.*

Proof. Let $\mu^\bullet$ abbreviate $\mu X.(E^{\bullet X})$. Define $\mathcal{R}$ to be the following relation

$$\left\{ (!E \,|\, G[\mathbf{0}], \prod_i \mu^\bullet \,|\, G[\mu^\bullet]) \;\middle|\; \begin{array}{l} G[X] \text{ is in } \mathrm{CCS}^m, \\ X \text{ is guarded in } G[X], \\ \text{and } i \geq 2 \end{array} \right\}.$$

We claim that $\mathcal{R}$ is a structural bisimulation up to $\sim$. Suppose that $G[\mu^\bullet]$ performs an action λ. Since X is guarded in $G[X]$, the action λ must be induced by $G[X] \xrightarrow{\lambda} G'[X]$. Clearly in this case the action

$$\prod_i \mu^\bullet \,|\, G[\mu^\bullet] \xrightarrow{\lambda} \prod_i \mu^\bullet \,|\, G'[\mu^\bullet]$$

is simulated by $!E \,|\, G[\mathbf{0}] \xrightarrow{\lambda} !E \,|\, G'[\mathbf{0}]$. Now suppose that $\mu^\bullet \xrightarrow{\lambda} H$. Since X does not appear free in E, the variable X must be guarded in $E^{\bullet X}$. It follows from $E^{\bullet X}[\mu^\bullet] \xrightarrow{\lambda} H$ that $E^{\bullet X} \xrightarrow{\lambda} H'$ and $H \equiv H'\{\mu^\bullet/X\}$ for some H'. By Lemma 23, one has that $E \xrightarrow{\lambda} H_1$ and $H' \sim X \,|\, H_1^{\bullet X}$ for some H_1 such that X is guarded in H_1. Consequently

$$\prod_i \mu^\bullet \,|\, G[\mu^\bullet] \xrightarrow{\lambda} \prod_i \mu^\bullet \,|\, H \,|\, G[\mu^\bullet] \sim \prod_{i+1} \mu^\bullet \,|\, H_1^{\bullet X}[\mu^\bullet] \,|\, G[\mu^\bullet],$$

which is simulated by

$$!E \,|\, G[\mathbf{0}] \xrightarrow{\lambda} !E \,|\, H_1 \,|\, G[\mathbf{0}] \sim !E \,|\, H_1^{\bullet X}[\mathbf{0}] \,|\, G[\mathbf{0}].$$

Due to the guardedness, the proofs of the other cases can be safely omitted. $\square$

Lemma 24 says that $!E$ can be equivalently presented in a more controlled manner. Instead of introducing an infinite copies of E, one could introduce two copies of $\mu X.(E^{\bullet X})$. The latter is more manageable than the former for the reason that X is guarded in $E^{\bullet X}$.

We mentioned before that when converting a process of CCS^m to a process of CCS^{gm}, an expression like $!Y$ has to be left intact until a binder μY props up. But when the binder μY does appear, how do we eliminate $!Y$ in favor of the guarded recursions? When unfolding $\mu Y.E$, the subexpression $!Y$ of E becomes $!\mu Y.E$. The first step could be to manipulate $!\mu Y.E$ into a shape that is considerably simpler.

Lemma 25. *If E is an expression in CCS^m, then $!\mu X.E \sim !E\{\mathbf{0}/X\}$.*

Proof. Generally $F \,|\, !\mu X.E \sim F \,|\, !E\{\mathbf{0}/X\}$ for every process expression F in CCS^m. An action of $!\mu X.E$ must take the form $!\mu X.E \xrightarrow{\lambda} E'[\mu X.E] \,|\, !\mu X.E$. So the action

$$F \,|\, !\mu X.E \xrightarrow{\lambda} F \,|\, E'[\mu X.E] \,|\, !\mu X.E \sim F \,|\, E'\{\mathbf{0}/X\} \,|\, !\mu X.E$$

is simulated by $F \,|\, !E\{\mathbf{0}/X\} \xrightarrow{\lambda} F \,|\, E'\{\mathbf{0}/X\} \,|\, !E\{\mathbf{0}/X\}$, where the equivalence is due to Lemma 22. □

Corollary 4. *$\mu X.E \sim \mu X.E\{(\mu X.E^{\circ X} \,|\, \mu X.E^{\circ X})/!X\}$ whenever E is in CCS^m and X is guarded in E.*

Proof. The following equivalences are due to Lemma 24 and Lemma 25:

$$\begin{aligned}
\mu X.E &\sim E\{\mu X.E/X\} \\
&\equiv E\{!\mu X.E/!X\}\{\mu X.E/X\} \\
&\sim E\{!E\{\mathbf{0}/X\}/!X\}\{\mu X.E/X\} \\
&\sim E\{(\mu X.E^{\circ X} \,|\, \mu X.E^{\circ X})/!X\}\{\mu X.E/X\}.
\end{aligned}$$

What the equivalences say is that $\mu X.E$ is a fixpoint of the equation

$$X \sim E\{(\mu X.E^{\circ X} \,|\, \mu X.E^{\circ X})/!X\}.$$

Since X is guarded in E, one could apply the fixpoint lemma (Lemma 10) to conclude that $\mu X.E \sim \mu X.E\{(\mu X.E^{\circ X} \,|\, \mu X.E^{\circ X})/!X\}$. □

Let's see an example. Suppose $E \equiv \mu X.(a.X + \overline{a}.!X)$. Then we have the following rewriting:

$$\begin{aligned}
E &\sim a.E + \overline{a}.!E \\
&\sim a.E + \overline{a}.!(a.\mathbf{0} + \overline{a}.!\mathbf{0}) \\
&\sim a.E + \overline{a}.!(a + \overline{a}) \\
&\sim a.E + \overline{a}.(\mu X.(a.X + \overline{a}.X) \,|\, \mu X.(a.X + \overline{a}.X)).
\end{aligned}$$

Now $E \sim \mu X.(a.X+\overline{a}.(\mu X.(a.X+\overline{a}.X) \,|\, \mu X.(a.X+\overline{a}.X)))$ follows from Lemma 10.

We are now in a position to prove the major lemma of this subsection.

Lemma 26. *$\mathrm{CCS}^m \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{gm}$.*

Proof. Let E be in CCS^m. We show by structural induction that E must be structurally bisimilar to some process expression in quasi-normal form:

$$(\prod_{h=1}^{k} (\underbrace{X_h \,|\, \ldots \,|\, X_h}_{i_h})) \,|\, (\prod_{g=1}^{p} !Y_g) \,|\, F. \tag{5}$$

Notice that Y_g is not guarded in $!Y_g$. But subexpressions of this form $!Y_g$ will be thrown away on the way of converting a process in CCS^m to a process in CCS^{gm}. The conversion is defined structurally by the following clauses.

- If E is a mixed choice, it is already in the form of (5).
- If $E \equiv E_1 \,|\, E_2$, then a simple application of the induction suffices. Both the conversions of E_1 and E_2 may contain $!Y$. The strong equivalence $!Y \sim\, !Y \,|\, !Y$ is necessary to replace the two occurrences of $!Y$ by $!Y$.
- If E is a restriction, then a simple application of the induction suffices.
- If $E \equiv \mu X.E_1$ then by induction E_1 must be structurally bisimilar to some quasi-normal form

  $$\prod_{h=1}^{k} (\underbrace{X_h \,|\, \ldots \,|\, X_h}_{i_h}) \,|\, \prod_{g=1}^{p} !Y_g \,|\, F_1.$$

  It follows by congruence that

  $$E \sim \mu X.(\prod_{h=1}^{k} (\underbrace{X_h \,|\, \ldots \,|\, X_h}_{i_h}) \,|\, \prod_{g=1}^{p} !Y_g \,|\, F_1).$$

  We are now converting

  $$\mu X.(\prod_{h=1}^{k} (\underbrace{X_h \,|\, \ldots \,|\, X_h}_{i_h}) \,|\, \prod_{g=1}^{p} !Y_g \,|\, F_1) \tag{6}$$

  into a structurally bisimilar quasi-normal form. The conversion of the fixpoint can be defined in three cases.

  - $X \notin \{X_1, \ldots, X_k, Y_1, \ldots, Y_p\}$. According to Lemma 21 and Lemma 22, the expression (6) is in this case structurally bisimilar to

    $$\prod_{h=1}^{k} (\underbrace{X_h \,|\, \ldots \,|\, X_h}_{i_h}) \,|\, \prod_{g=1}^{p} !Y_g \,|\, \mu X.F_1^{\circ}$$

    where F_1° is $F_1\{\prod_{h=1}^{k} (\underbrace{X_h \,|\, \ldots \,|\, X_h}_{i_h}) \,|\, X/X\}$. Since every free process variable in F_1 is guarded, every free process variable, as well as X, in $\mu X.F_1^{\circ}$ must also be guarded. Now using the following equivalence

    $$!(V_1 \,|\, \ldots \,|\, V_m) \sim\, !V_1 \,|\, \ldots \,|\, !V_m$$

    stated in Lemma 19 to convert the expression so that the induced operator "!" applies only to individual variables. Moreover we may apply Corollary 4 to remove all the occurrences of $!X$. The result is a quasi-normal form $F_1^{\diamond}$ structurally bisimilar to $\mu X.F_1^{\circ}$. So in this case (6) is structurally bisimilar to

    $$\prod_{h=1}^{k} (\underbrace{X_h \,|\, \ldots \,|\, X_h}_{i_h}) \,|\, \prod_{g=1}^{p} !Y_g \,|\, F_1^{\diamond}. \tag{7}$$

– $X = X_1$. By using Corollary 1, Corollary 3, Lemma 20 and Lemma 19, the expression (6) can be converted to

$$\prod_{h=2}^{k} !X_h \mid \prod_{g=1}^{p} !Y_g \mid \mu X.(X \mid F_1\{\mathbf{0}/X\}). \tag{8}$$

By Lemma 24, the expression (8) is structurally bisimilar to

$$\prod_{h=2}^{k} !X_h \mid \prod_{g=1}^{p} !Y_g \mid F_1^{\diamond} \tag{9}$$

where $F_1^{\diamond}$ is the process expression $\mu X.F_1^{\circ X} \mid \mu X.F_1^{\circ X}$.

– $X = Y_1$. According to Corollary 2, Corollary 3, Lemma 20, Lemma 19 and Lemma 24, the expression (6) is structurally bisimilar to

$$\prod_{h=1}^{k} !X_h \mid \prod_{g=2}^{p} !Y_g \mid F_1^{\diamond} \tag{10}$$

where $F_1^{\diamond}$ is the process expression $\mu X.F_1^{\circ X} \mid \mu X.F_1^{\circ X}$.

It is clear from (7), (9) and (10) that E is structurally bisimilar to a quasi-normal form. In (9) and (10) the bound variable X does not occur in any subterm of the form $!X$. In (7) however such occurrence is possible. But according to Corollary 4, these occurrences can be eliminated.

Now given a process P in CCS^m, the structural conversion defined above allows us to find a process P' that is structurally bisimilar to P. The process P' could contain $!\mathbf{0}$, which can be replaced by $\mathbf{0}$ of course. Therefore P is structurally bisimilar to a guarded process in CCS^{gm}. □

The proof of Lemma 26 actually provides an effective procedure to convert a CCS^m process to a CCS^{gm} process. Let's see an example. Suppose $F \equiv Z \mid !X \mid !Y \mid (a.X + \overline{a}.!X)$ and $E \equiv \mu X.F$. We may apply the following rewriting:

$$\begin{array}{rcl}
E & \equiv & \mu X.(Z \mid !X \mid !Y \mid (a.X + \overline{a}.!X)) \\
& \sim & \mu X.(X \mid Z \mid !Y \mid (a.X + \overline{a}.!X)) \\
& \sim & !(Z \mid !Y) \mid \mu X.(X \mid (a.X + \overline{a}.!X)) \\
& \sim & !Z \mid !Y \mid \mu X.(X \mid (a.X + \overline{a}.!X)) \\
& \sim & !Z \mid !Y \mid \mu X.(X \mid (a.\mathbf{0} + \overline{a}.!\mathbf{0})) \\
& \sim & !Z \mid !Y \mid \mu X.(X \mid (a + \overline{a})) \\
& \sim & !Z \mid !Y \mid \mu X.(a.X + \overline{a}.X) \mid \mu X.(a.X + \overline{a}.X).
\end{array}$$

The component $\mu X.(a.X + \overline{a}.X) \mid \mu X.(a.X + \overline{a}.X)$ is in CCS^{gm}. When applying our conversion, the expression E is typically sitting inside a process in which Y, Z are bound. Further conversions would deal with the components $!Z$, $!Y$.

Corollary 5. *There is an algorithm $\mathcal{I}^{gm}$ such that, for each* CCS^m *process P, $\mathcal{I}^{gm}(P)$ returns a* CCS^{gm} *process that is structurally bisimilar to P.*

The bulk of this section is devoted to dealing with the choice operator. Without the choice operator Corollary 5 is much easier to prove. See [9] for a proof of a similar result in Ambient Calculus which is of course choice free.

The above exercise suggests that the recursion semantics of CCS^m can be alternatively defined by the following rule.

$$\frac{E\{E/X\} \xrightarrow{\lambda} E'}{\mu X.E \xrightarrow{\lambda} E'\{\mu X.E/X\}}$$

It gives rise to a simpler semantics that enjoys the finite branching property.

2.4 First Order Recursion

The result in Section 2.3.2 suggests to introduce the replication processes of the form $!P$ at the syntactic level. This is what Milner did in [44], which was actually motivated by a similar operation in linear logic [24]. Milner has defined the semantics of the replicator by the following rule.

$$\frac{P \,|\, !P \xrightarrow{\lambda} P'}{!P \xrightarrow{\lambda} P'}$$

The advantage of this rule is that it makes obvious the validity of the equivalence $!P \sim P \,|\, !P$. Intuitively $!P$ is the same as $P \,|\, !P$. In other words $!P$ provides a potentially infinite number of copies of P. However the above rule introduces infinite branching at the syntactical level, just like the unguarded recursion in CCS^m. One way to retain the finite branching property is to use the following rule.

$$\frac{P \,|\, P \xrightarrow{\lambda} P'}{!P \xrightarrow{\lambda} P' \,|\, !P}$$

One may also use the following rules due to Busi, Gabbrielli and Zavattaro [7].

$$\frac{P \xrightarrow{\lambda} P'}{!P \xrightarrow{\lambda} P' \,|\, !P} \qquad \frac{P \xrightarrow{a} P' \quad P \xrightarrow{\overline{a}} P''}{!P \xrightarrow{\tau} P' \,|\, P'' \,|\, !P}$$

It is easy to see that the three formulations of the replication are equivalent.

Let CCS$^!$, respectively CCS$^{m!}$ and CCS$^{s!}$, denote the variant of CCS, respectively CCS^m and CCS^s, whose infinite behaviors are defined by the replicator. Our agenda in the present subsection is to work out the relative expressive power of these variants.

One obvious advantage of using the replication instead of the fixpoint operation is that one could give a purely first order presentation of CCS. There is no need for process variables. This is why the above rules are defined on the set of processes, not on the set of expressions.

Let $depth_{()}(\_)$ be the function that returns the maximum number of nested localization operations. It can be inductively defined as follows.

$$\begin{aligned}
depth_{()}(\mathbf{0}) &= 0, \\
depth_{()}(\lambda.P) &= depth_{()}(P), \\
depth_{()}(P \,|\, P') &= max\{depth_{()}(P), depth_{()}(P')\}, \\
depth_{()}((a)P) &= depth_{()}(P) + 1, \\
depth_{()}(P{+}P') &= max\{depth_{()}(P), depth_{()}(P')\}, \\
depth_{()}(!P) &= depth_{()}(P).
\end{aligned}$$

A useful property of the replication is that the depth of the nested localization operations in a process remain fixed over process evolution.

Lemma 27. *Suppose P is in* $\mathrm{CCS}^!$*. If $P \xrightarrow{\lambda} P'$ then $depth_{()}(P') = depth_{()}(P)$.*

By Lemma 27, new nesting of localization operations is never created dynamically, rendering α-conversion unnecessary.

2.4.1 Expansion Property of Replication

First let's take a look at a simple but illustrating example. Consider an arbitrary infinite action sequence of the process $A \stackrel{\mathrm{def}}{=} \overline{b}.a \,|\, !\overline{a}.b \,|\, !\overline{b}.a$, say

$$A \xrightarrow{\overline{b}} a \,|\, !\overline{a}.b \,|\, !\overline{b}.a \xrightarrow{\tau} b \,|\, !\overline{a}.b \,|\, !\overline{b}.a \xrightarrow{\tau} !\overline{a}.b \,|\, a \,|\, !\overline{b}.a \xrightarrow{\tau} \ldots .$$

There appears some kind of periodicity in the infinite number of processes. Both $a \,|\, !\overline{a}.b \,|\, !\overline{b}.a$ and $!\overline{a}.b \,|\, a \,|\, !\overline{b}.a$ have the same set of concurrent components $\{a, !\overline{a}.b, !\overline{b}.a\}$. Obviously $!\overline{a}.b \,|\, a \,|\, !\overline{b}.a$ can do whatever $a \,|\, !\overline{a}.b \,|\, !\overline{b}.a$ can do. This is a special case of a general phenomenon in $\mathrm{CCS}^!$: For an infinite action sequence

$$A_0 \xrightarrow{\lambda_0} A_1 \xrightarrow{\lambda_1} A_2 \xrightarrow{\lambda_2} \ldots$$

in $\mathrm{CCS}^!$, there must exist two processes A_i, A_j, where $i < j$, such that A_j contains everything A_i has. In other words, A_j can do whatever A_i can do. This interesting property of the replication was first pointed out by Busi, Gabbrielli and Zavattaro in [7]. In the terminology of order theory, it says that the set of the derivatives of a process of $\mathrm{CCS}^!$ is a well quasi-order. This property is of crucial importance to some of the counter examples given later on in the paper. Busi and Zavattaro have provided in [9] a detailed proof that an expansion order defined for certain ambients is a well quasi-order. For the benefit of the reader, below we give a self-contained account of this property.

In the rest of the subsection we introduce a syntactic order on the $\mathrm{CCS}^!$-processes. This syntactic order formalizes the intuition that one process contains every concurrent component of another. The adequacy of the formalization is supported by the Compatibility Lemma. We prove the Expansion Lemma saying that the set of the derivatives of a $\mathrm{CCS}^!$-process is a well quasi-order under this order relation. It then follows that every infinite action sequence contains an infinite chain of processes with increasing interactive capacities.

Let's start with something in order theory [13, 37].

Definition 9. *A quasi-order $(X, \leq)$ is a binary relation $\leq$ that is reflexive and transitive.*

An important concept that is most relevant to us is the well quasi-order structure [37].

Definition 10. *A* well quasi-order *$(X, \leq)$ is a quasi-order such that, for any infinite sequence $x_0, x_1, x_2, \ldots$ in X, there exist indexes $i < j$ with $x_i \leq x_j$.*

The main reason that a well quasi-order is an interesting structure is the following property.

Lemma 28 (Erdös and Rado)**.** *Suppose $(X, \leq)$ is a well quasi-order. An infinite sequence $x_0, x_1, x_2, \ldots$ in X must contain an infinite increasing subsequence $x_{i_0} \leq x_{i_1} \leq x_{i_2} \leq \ldots$, where $i_0 < i_1 < i_2 < \ldots$.*

Proof. Suppose $x_0, x_1, x_2, \ldots$ is an infinite sequence in X. Consider the set $\{x_i \mid \forall j > i. x_i \not\leq x_j\}$. By the definition of the well quasi-order, this set cannot be infinite. Then an element beyond all the elements of this set starts an infinite increasing sequence. $\square$

Finkel and Schnoebelen have surveyed in [17] some applications of the well quasi-order structure in theoretical computer science.

We are going to define a quasi-order on $\mathrm{CCS}^!$ processes, which turns out to be a well quasi-order. To simplify the account, we make two syntactical assumptions in the rest of this subsection. First of all we assume that all the local names are distinct from all the global names in consideration. Moreover α-conversion is *not* applied when a process replicates part of itself. This is harmless in view of the remark after Lemma 27. Secondly we identify syntactically two processes up to the symmetry and associativity of the compositions. We write $A = B$ if B can be rewritten from A, or vice versa, by applying the following equalities

$$\begin{aligned} P \,|\, Q &= Q \,|\, P, & (11)\\ P \,|\, (Q \,|\, R) &= (P \,|\, Q) \,|\, R. & (12) \end{aligned}$$

The proof of the following lemma is a routine induction on the number of rewriting steps.

Lemma 29. *If $P = Q \xrightarrow{\lambda} Q'$ then $P \xrightarrow{\lambda} P' = Q'$ for some P'.*

The equalities (11) and (12) can be applied to convert every $\mathrm{CCS}^!$ process to a normal form.

Definition 11. *Suppose that P is a* CCS$^!$ *process. A* concurrent normal form *of P is some*

$$\prod_{i\in I} P_i$$

such that $P = \prod_{i\in I} P_i$ and, for each $i \in I$, P_i is not a composition. We say that P_i, for each $i \in I$, is a concurrent component *of P.*

The terminology "normal forms" usually implies uniqueness up to some equivalence. Hence the following lemma.

Lemma 30. *The concurrent normal form of a* CCS$^!$ *process is unique up to $=$.*

By the above definition, in the following process

$$\lambda_1.A \,|\, !R \,|\, (c)(\lambda_2.B \,|\, !S) \,|\, \lambda_3.\lambda_4.(a.C \,|\, b.D) \tag{13}$$

there are four concurrent components. They are $\lambda_1.A$, $!R$, $(c)(\lambda_2.B \,|\, !S)$ and $\lambda_3.\lambda_4.(a.C \,|\, b.D)$ respectively. If the process (13) performs the λ_3 action, then $\lambda_4.(a.C \,|\, b.D)$ becomes a concurrent component. If the process continues to do the λ_4-action, then the concurrent component $\lambda_4.(a.C \,|\, b.D)$ splits to two concurrent components $a.C$, $b.D$. Similarly $!R$ may induce an action and produce new concurrent components. However there are only a finite number of syntactically distinct concurrent components ever created since every concurrent component is a sub-term of the process (13). This example leads to the following definition [8].

Definition 12. *The set of* concurrent subexpressions *of A, denoted by $Csub(A)$, is defined by the following structural induction.*

$$\begin{aligned}
Csub(\mathbf{0}) &\stackrel{\text{def}}{=} \{\mathbf{0}\},\\
Csub(\lambda.A') &\stackrel{\text{def}}{=} \{\lambda.A'\} \cup Csub(A'),\\
Csub(A' \,|\, A'') &\stackrel{\text{def}}{=} Csub(A') \cup Csub(A''),\\
Csub(A'+A'') &\stackrel{\text{def}}{=} \{A'+A''\} \cup Csub(A') \cup Csub(A''),\\
Csub((a)A') &\stackrel{\text{def}}{=} Csub(A'),\\
Csub(!A') &\stackrel{\text{def}}{=} \{!A'\} \cup Csub(A').
\end{aligned}$$

A concurrent subexpression is neither in composition form nor in localization form. It is intuitively clear that $Csub(A)$ is finite for each process. To appreciate the next lemma, it is important to bear in mind that α-conversion is not used when a subexpression is duplicated by an action.

Lemma 31. *For each process P of* CCS$^!$*, $Csub(P)$ is finite; and moreover $\bigcup_{P' \in \mathfrak{Drv}(P)} Csub(P') = Csub(P)$.*

Lemma 31 gives us a hint on how one might get a well quasi-order: One could introduce an order relation that compares the sets of the occurrences of the concurrent subexpressions in a structural way. This important syntactic order relation were first proposed by Busi, Gabbrielli and Zavattaro [7].

Definition 13. *The* syntactical expansion $\preceq$ *on* $\mathrm{CCS}^!$ *processes is defined inductively as follows:*

- $P \preceq P$*;*
- $P \preceq Q$ *whenever* $Q = P \,|\, R$*;*
- $(c)P \preceq (c)Q$ *whenever* $P \preceq Q$*;*
- $P \preceq Q$ *whenever* $P = P_0 \,|\, P_1$*,* $Q = Q_0 \,|\, Q_1$*,* $P_0 \preceq Q_0$ *and* $P_1 \preceq Q_1$*.*

Notice that in our approach we do not have $\mathbf{0} \,|\, P \preceq P$ since we do not equate $\mathbf{0} \,|\, P$ with P syntactically. The absence of $\mathbf{0} \,|\, P = P$ simplifies some of the arguments below. The next lemma for example follows immediately from the definition.

Lemma 32. *If* $(c)P \preceq A$ *and* A *is not a composition then there exists some* Q *such that* $A \equiv (c)Q$*.*

The following is a useful lemma about the relation $\not\preceq$ that allows one to reduce the size of the processes being compared.

Lemma 33. *Suppose* $P_0 \,|\, P_1 \not\preceq Q_0 \,|\, Q_1$ *for processes* P_0, P_1, Q_0, Q_1 *in* $\mathrm{CCS}^!$*. If* $P_1 \preceq Q_1$ *then* $P_0 \not\preceq Q_0$*.*

The expansion order is transitive. Hence the following lemma.

Lemma 34. *For each process* P *of* $\mathrm{CCS}^!$*,* $(\mathfrak{Drv}(P), \preceq)$ *is a quasi-order.*

Intuitively $P \preceq Q$ means that Q contains at least as many possible individual processes running concurrently as P. The property described in the next lemma is called compatibility in [17].

Lemma 35 (Compatibility Lemma)**.** *Suppose that* P, Q *are* $\mathrm{CCS}^!$ *processes. If* $P \preceq Q$ *and* $P \xrightarrow{\lambda} P'$ *then* Q' *exists such that* $Q \xrightarrow{\lambda} Q'$ *and* $P' \preceq Q'$*.*

Proof. The lemma can be proved by a simple induction on the depth of the nested localization operations. □

We have done all the preparations for the main lemma of this subsection. The Expansion Lemma proved below is one of the key technical lemmas in classifying the expressive power of CCS variants.

Lemma 36 (Expansion Lemma)**.** *For each process* P *of* $\mathrm{CCS}^!$*,* $(\mathfrak{Drv}(P), \preceq)$ *is a well quasi-order.*

Proof. Recall that $\mathfrak{Drv}(P)$ is the set of all the derivatives of P, including P itself. We will prove the lemma by contradiction. Assume that in $\mathrm{CCS}^!$ there were an infinite sequence of processes

$$A_0, A_1, \ldots, A_p, \ldots \tag{14}$$

such that $\forall m, n.m<n \Rightarrow A_m \not\preceq A_n$. For each $p \geq 0$, let

$$\prod_{i \in I_p} A_p^i$$

be the concurrent normal form of A_p. Now consider the concurrent components of $\prod_{i \in I_0} A_0^i$. There are two possibilities:

- I_0 is the singleton set $\{*\}$. Then $\prod_{i \in I_0} A_0^i$ contains a single concurrent component A_0^*. By the assumption the following proposition holds

  $$\forall p>0.\forall i \in I_p.A_0^* \not\preceq A_p^i.$$

  Let B_0 be A_0^* in this case. We then continue to examine the infinite sequence $A_2, A_3, \ldots, A_p, \ldots$.

- The size of I_0 is greater than 1. Let $A_0^{i_0}$ be a concurrent component of $\prod_{i \in I_0} A_0^i$. If $A_0^{i_0}$ leads an infinite increasing sequence

  $$A_0^{i_0} \preceq A_{p_1}^{i_{p_1}} \preceq \ldots \preceq A_{p_k}^{i_{p_k}} \preceq \ldots \quad (15)$$

  where $p_1 < p_2 < \ldots < p_k < \ldots$ and $A_{p_k}^{i_{p_k}}$ is a concurrent component of $\prod_{i \in I_{p_k}} A_{p_k}^i$, then according to Lemma 33 we may continue to examine the infinite sequence

  $$\prod_{i \in I_0 \setminus \{i_0\}} A_0^i, \prod_{i \in I_{p_1} \setminus \{i_{p_1}\}} A_{p_1}^i, \ldots, \prod_{i \in I_{p_k} \setminus \{i_{p_k}\}} A_{p_k}^i, \ldots.$$

  Notice that the number of the concurrent components of $\prod_{i \in I_0 \setminus \{i_0\}} A_0^i$ is smaller than that of $\prod_{i \in I_0} A_0^i$.

  If $A_0^{i_0}$ does not lead any infinite increasing sequence like (15), then there must exist some $q > 0$ and some concurrent component $A_q^{i_q}$ of $\prod_{i \in I_q} A_q^i$ such that the following proposition holds

  $$\forall p>q.\forall i \in I_p.A_q^{i_q} \not\preceq A_p^i.$$

  Let B_0 be $A_q^{i_q}$ in this case. We then continue to examine the infinite sequence

  $$A_{q+1}, A_{q+2}, \ldots, A_{q+k}, \ldots.$$

By the above construction we eventually obtain an infinite sequence

$$B_0, B_1, \ldots, B_k, \ldots$$

such that $B_{k'} \not\preceq B_{k''}$ whenever $0 \leq k' < k''$ and B_k contains a single concurrent component for each $k \geq 0$. In view of Lemma 31 and Lemma 32, we can subtract from this sequence an infinite subsequence

$$B_{r_0}, B_{r_1}, \ldots, B_{r_k}, \ldots$$

that satisfies either of the following conditions.

1. B_{r_k} is not in localization form for each $k \geq 0$.
2. There exists some d such that $B_{r_k} \equiv (d)B'_{r_k}$ for all $k \geq 0$.

The first case would lead to a contradiction by Lemma 31. The second case would eventually lead to the same contradiction by Lemma 32 and Lemma 27 since we can repeat the above construction to $B'_{r_0}, B'_{r_1}, \ldots, B'_{r_k}, \ldots$. □

Lemma 36 and Lemma 28 immediately imply the following proposition.

Proposition 4. *Suppose that P is a process in* $\mathrm{CCS}^!$. *If $P_0, P_1, P_2, \ldots$ is an infinite sequence in $(\mathfrak{Drv}(P), \preceq)$, then there is an infinite increasing subsequence $P_{i_0} \preceq P_{i_1} \preceq P_{i_2} \preceq \ldots$, where $i_0 < i_1 < i_2 < \ldots$.*

Let's say that an infinite action sequence

$$A_0 \xrightarrow{\lambda_0} A_1 \xrightarrow{\lambda_1} A_2 \xrightarrow{\lambda_2} \ldots$$

in $\mathrm{CCS}^!$ is *well quasi-ordered* if there is an infinite subsequence satisfying

$$A_{i_0} \preceq A_{i_1} \preceq A_{i_2} \preceq \ldots$$

where $i_0 < i_1 < i_2 < \ldots$. Proposition 4 says that every infinite action sequence in $\mathrm{CCS}^!$ is well quasi-ordered.

2.4.2 Replication and Mixed Choice

The relationship between the replicator and the fixpoint operator is obvious when both choice and recursion are guarded.

Fact 1 (Milner [44]; Giambiagi, Schneider, Valencia [27]). $\mathrm{CCS}^! \sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}$ *and* $\mathrm{CCS}^{gm} \sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{m!}$.

Proof. Intuitively $!P$ of $\mathrm{CCS}^!$ can be interpreted in CCS as $\mu X.(P \,|\, X)$. Conversely an encoding $\mathcal{I}^!$ from CCS^{gm} to $\mathrm{CCS}^{m!}$ is homomorphic on the non-fixpoint operators. It maps $\mu X.E$ onto $(d)(E\{\overline{d}/X\} \,|\, !d.E\{\overline{d}/X\})$, where d does not occur in E. □

The above encoding provides an algorithm $\mathcal{I}^!$ that converts a CCS^{gm} process P to a $\mathrm{CCS}^!$ process $\mathcal{I}^!(P)$ that is bisimilar to P. The composition $\mathcal{I}^!\mathcal{I}^{gm}$ is an algorithm that converts a CCS^m process to a $\mathrm{CCS}^!$ process.

There are two points worth making about the encoding from CCS^{gm} to $\mathrm{CCS}^{m!}$. Firstly the encoding does not work for CCS. The process $b + \mu X.(a \,|\, X)$ would be encoded by $b + (d)(\overline{d} \,|\, !d.(a \,|\, \overline{d}))$, which has to be wrong since it has the preemptive power to get rid of a. Secondly the encoding works for CCS^m, but it does not provide a termination preserving encoding. The interpretation of the terminating process $\mu X.(a \,|\, X)$ for instance is the divergent process $(d)(\overline{d} \,|\, !d.(a \,|\, \overline{d}))$.

The result stated in Fact 1 can be strengthened significantly. In one direction we have the following lemma.

Lemma 37. $\mathrm{CCS}^{!} \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{gm}$.

Proof. In view of Lemma 26, it is enough to define an encoding $[\![\_]\!]$ from $\mathrm{CCS}^{!}$ to CCS^{m}. For every process P of $\mathrm{CCS}^{!}$, the translation $[\![P]\!]$ is a guarded choice definable in CCS^{m}. The encoding $[\![\_]\!]$ is defined structurally:

- $P \equiv \mathbf{0}$. In this case, $[\![\mathbf{0}]\!] \overset{\mathrm{def}}{=} \mathbf{0}$.
- $P \equiv \lambda.P'$. In this case, $[\![P]\!] \overset{\mathrm{def}}{=} \lambda.[\![P']\!]$.
- $P \equiv P_1 \,|\, P_2$. By induction $[\![P_1]\!]$ is of the form $\sum_{i\in I_1} \lambda_i.Q_i$ and $[\![P_2]\!]$ is of the form $\sum_{i\in I_2} \lambda_i.Q_i$ such that $I_1 \cap I_2 = \emptyset$. Let $[\![P]\!]$ be

$$\sum_{i\in I_1} \lambda_i.(Q_i|[\![P_2]\!]) + \sum_{i\in I_2} \lambda_i.([\![P_1]\!]|Q_i) + \sum_{i\in I_1, j\in I_2}^{\overline{\lambda_i}=\lambda_j} \tau.(Q_i|Q_j)$$

  obtained by applying the expansion law.
- $P \equiv (c)P'$. By induction $[\![P']\!]$ is of the form $\sum_{i\in I} \lambda_i.Q_i$. Let $[\![P]\!]$ be $\sum_{i\in I'} \lambda_i.(c)Q_i$, where $I' = \{i \mid i \in I \wedge n(\lambda_i) \neq c\}$. When I' is empty, $[\![P]\!]$ is simply $\mathbf{0}$.
- $P \equiv P_1{+}P_2$. By induction $[\![P_1]\!]$ is of the form $\sum_{i\in I_1} \lambda_i.Q_i$ and $[\![P_2]\!]$ is of the form $\sum_{i\in I_2} \lambda_i.Q_i$ such that $I_1 \cap I_2 = \emptyset$. Let $[\![P]\!]$ be $\sum_{i\in I_1\cup I_2} \lambda_i.Q_i$.
- $P \equiv\, !P'$. By induction $[\![P']\!]$ is of the form $\sum_{i\in I} \lambda_i.Q_i$. If $I = \emptyset$ then let $[\![P]\!]$ be $\mathbf{0}$; otherwise let $[\![P]\!]$ be the summation

$$\sum_{i\in I} \lambda_i.(Q_i \,|\, Z_P) + \sum_{i,j\in I}^{\lambda_i=\overline{\lambda_j}} \tau.(Q_i \,|\, Q_j \,|\, Z_P),$$

  where Z_P is the following fixpoint

$$\mu Z.(\sum_{i\in I} \lambda_i.(Q_i \,|\, Z) + \sum_{i,j\in I}^{\lambda_i=\overline{\lambda_j}} \tau.(Q_i \,|\, Q_j \,|\, Z)).$$

The encoding clearly gives rise to a termination preserving subbisimilarity from $\mathrm{CCS}^{!}$ to CCS^{m}. □

The following result appears to be new. It summarizes the relationships between several variants of CCS.

Corollary 6. $\mathrm{CCS}^{!} \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{m} \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{gm} \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{m!} \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{!}$.

A proof of the following fact can be read off from the proof of the above corollary.

Corollary 7. $\mathrm{CCS}^s \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{s!}$.

It is interesting to observe that Corollary 6 implies that the difference between CCS and CCS^m is the same as that between CCS and $\mathrm{CCS}^!$. The former is due to the variations on the choice and the latter is attributed to the two forms of recursion.

An important consequence of Corollary 6 is that we could introduce the expansion order for CCS^m and CCS^{gm} so that Compatibility Property (Lemma 35) and Expansion Property (Proposition 4) are valid for these calculi. The expansion order for CCS^{gm} is defined in terms of the expansion order $\preceq$ for $\mathrm{CCS}^!$.

For CCS^{gm} processes P, Q, $P \preceq^{gm} Q$ if and only if $\mathcal{I}^!(P) \preceq \mathcal{I}^!(Q)$.

The quasi-well order structure is an immediate consequence of this definition.

Proposition 5. *Suppose P is a process in* CCS^{gm}. *If $P_0, P_1, P_2, \ldots$ is an infinite sequence in $(\mathfrak{Drv}(P), \preceq^{gm})$, then there is an infinite increasing subsequence $P_{i_0} \preceq^{gm} P_{i_1} \preceq^{gm} P_{i_2} \preceq^{gm} \ldots$, where $i_0 < i_1 < i_2 < \ldots$.*

Proof. Suppose $P_0, P_1, P_2, \ldots$ is an infinite sequence in CCS^{gm}. Then in $\mathrm{CCS}^!$ one has an infinite sequence $\mathcal{I}^!(P_0), \mathcal{I}^!(P_1), \mathcal{I}^!(P_2), \ldots$. By Proposition 4, there is an infinite increasing subsequence $\mathcal{I}^!(P_{i_0}) \preceq \mathcal{I}^!(P_{i_1}) \preceq \mathcal{I}^!(P_{i_2}) \preceq \ldots$, where $i_0 < i_1 < i_2 < \ldots$. It follows that $P_{i_0} \preceq^{gm} P_{i_1} \preceq^{gm} P_{i_2} \preceq^{gm} \ldots$. □

The compatibility of $\preceq^{gm}$ also holds.

Lemma 38. *Suppose P, Q are processes in* CCS^{gm}. *If $P \preceq^{gm} Q$ and $P \xrightarrow{\lambda} P'$ then Q' exists such that $Q \xrightarrow{\lambda} Q'$ and $P' \preceq^{gm} Q'$.*

Proof. Suppose $A \preceq^{gm} B$ and $A \xrightarrow{\lambda} A'$. By definition $\mathcal{I}^!(A) \preceq \mathcal{I}^!(B)$. The encoding $\mathcal{I}^!$ defined in the proof of Fact 1 satisfies the following property:

If $A \xrightarrow{\lambda} A'$ then $\mathcal{I}^!(A) \Longrightarrow\xrightarrow{\lambda}= \mathcal{I}^!(A') \mid \prod_i \mathbf{0}$ for some $i \geq 0$.

It follows from $\mathcal{I}^!(A) \preceq \mathcal{I}^!(B)$ that $\mathcal{I}^!(B) \Longrightarrow\xrightarrow{\lambda} C$ and $\mathcal{I}^!(A') \mid \prod_i \mathbf{0} \preceq C$. By the definition of $\preceq$, the process $\mathcal{I}^!(B)$ literally contains the process $\mathcal{I}^!(A)$. So it must be the case that $C = \mathcal{I}^!(B') \mid \prod_i \mathbf{0}$ and that $B \xrightarrow{\lambda} B'$. But then it is easy to see that $\mathcal{I}^!(A') \preceq \mathcal{I}^!(B')$. Hence $A' \preceq^m B'$. □

Now we may define an expansion order $\preceq^m$ on CCS^m processes in terms of $\preceq^{gm}$.

For CCS^m processes P, Q, $P \preceq^m Q$ if and only if $\mathcal{I}^{gm}(P) \preceq \mathcal{I}^{gm}(Q)$.

It is obvious that $\preceq^{gm}$ satisfies Expansion Property.

Proposition 6. *Suppose P is a process in* CCS^m. *If $P_0, P_1, P_2, \ldots$ is an infinite sequence in $(\mathfrak{Drv}(P), \preceq^m)$, then there is an infinite increasing subsequence $P_{i_0} \preceq^m P_{i_1} \preceq^m P_{i_2} \preceq^m \ldots$, where $i_0 < i_1 < i_2 < \ldots$.*

The structural bisimilarity between P and $\mathcal{I}^{gm}(P)$ forces $\preceq^m$ to inherit all the good properties from $\preceq^{gm}$. In particular CCS^m satisfies Compatibility Property.

Lemma 39. *Suppose P, Q are processes in* CCS^m. *If $P \preceq^m Q$ and $P \xrightarrow{\lambda} P'$ then Q' exists such that $Q \xrightarrow{\lambda} Q'$ and $P' \preceq^m Q'$.*

2.4.3 Replication, or Fixpoint

We have seen in Proposition 3 that there is no termination preserving subbisimilarity from CCS to CCS$^!$. Using Expansion Property proved in Section 2.4.1, we can strengthen that result significantly.

Proposition 7. CCS $\not\sqsubseteq^{ccs}$ CCS$^!$.

Proof. We show that *RanTimer* cannot be bisimulated by any process in CCS$^!$. Assume that a process A of CCS$^!$ bisimulated *RanTimer*. To bisimulate the action

$$\mathit{RanTimer} \xrightarrow{s} \mathbf{0}$$

there must exist some A_0, A_0', A_0'' such that

$$A \Longrightarrow A_0 \xrightarrow{s} A_0' \Longrightarrow A_0'' \approx \mathbf{0}.$$

Clearly $A_0 \approx \mathit{RanTimer}$ and $A_0' \approx A_0'' \approx \mathbf{0}$. Now A_0 should bisimulate $\mathit{RanTimer} \xrightarrow{s} \mathit{tick} \,|\, \mathbf{0}$. Therefore some A_1, A_1', A_1'' exist such that

$$A_0 \Longrightarrow A_1 \xrightarrow{s} A_1' \Longrightarrow A_1'' \approx \mathit{tick} \,|\, \mathbf{0}.$$

Again it is clear that $A_1 \approx \mathit{RanTimer}$. Now if $A_0 \Longrightarrow A_1 \xrightarrow{s} A_1'$ is bisimulated by $\mathit{RanTimer} \xrightarrow{s} Q$, then $A_1'' \approx Q$ since Q cannot do any tau action. Hence $A_1' \approx A_1'' \approx \mathit{tick} \,|\, \mathbf{0}$. In this way we may construct the following infinite sequence

$$A_0 \Longrightarrow A_1 \Longrightarrow \ldots \Longrightarrow A_i \Longrightarrow \ldots$$

such that

$$A_i \xrightarrow{s} \underbrace{\overset{tick}{\Longrightarrow} \cdots \overset{tick}{\Longrightarrow}}_{i \text{ times}} \approx \mathbf{0}$$

for each $i \geq 0$. Now we may apply Proposition 4 to get an infinite increasing subsequence

$$A_{i_0} \preceq A_{i_1} \preceq \ldots \preceq A_{i_p} \preceq \ldots.$$

By construction,

$$A_{i_0} \xrightarrow{s} \underbrace{\overset{tick}{\Longrightarrow} \cdots \overset{tick}{\Longrightarrow}}_{i_0 \text{ times}} \approx \mathbf{0} \tag{16}$$

and

$$A_{i_1} \xrightarrow{s} \underbrace{\overset{tick}{\Longrightarrow} \cdots \overset{tick}{\Longrightarrow}}_{i_1 \text{ times}} \approx \mathbf{0}. \tag{17}$$

It follows from Lemma 35 and (16) that

$$A_{i_1} \xrightarrow{s} \underbrace{\overset{tick}{\Longrightarrow} \cdots \overset{tick}{\Longrightarrow}}_{i_0 \text{ times}} A' \tag{18}$$

for some A'. If $A' \not\approx \mathbf{0}$ then the only actions A' may perform are *tick* action. It is impossible for A' to perform an infinite sequence of *tick* action since *RanTimer* cannot do an infinite sequence of *tick* action after an s-action. So we must have

$$A' \underbrace{\overset{tick}{\Longrightarrow} \cdots \overset{tick}{\Longrightarrow}}_{k \text{ times}} \approx \mathbf{0}$$

for some $k > 0$. According to Definition 13 it is easy to prove by structural induction that

$$A_{i_1} \underbrace{\overset{tick}{\Longrightarrow} \cdots \overset{tick}{\Longrightarrow}}_{k \text{ times}} A'' \xrightarrow{s} \underbrace{\overset{tick}{\Longrightarrow} \cdots \overset{tick}{\Longrightarrow}}_{i_0 \text{ times}} \approx \mathbf{0} \tag{19}$$

for some A''. Since $k > 0$, there must exist some A''' such that

$$A_{i_1} \Longrightarrow A''' \xrightarrow{tick} .$$

Now A''' would have to bisimulate *RanTimer* because A_{i_1} bisimulates the random timer. This is a contradiction since A''' would not be able to simulate the action *RanTimer* $\xrightarrow{s} \mathbf{0}$. It follows that (18) must be in the following form

$$A_{i_1} \xrightarrow{s} \underbrace{\overset{tick}{\Longrightarrow} \cdots \overset{tick}{\Longrightarrow}}_{i_0 \text{ times}} \approx \mathbf{0}. \tag{20}$$

The action sequences of (17) and (20) must be induced by two distinct s-prefixes, which must be the summands of a same mixed choice. Otherwise A_{i_1} would be able to do two consecutive s-actions.

The above argument can be repeated to arrive at the following conclusion: For each $p \geq 0$, A_{i_p} contains a mixed choice with at least $p+1$ summands. But this contradicts Lemma 31. □

2.5 Recursive Definition

The infinite behaviors can also be admitted through *constant definition* [39]. We shall spend a minute to explain the mechanism of constant definition. Suppose $E_1, \ldots, E_n$ are expressions with free process variables $X_1, \ldots, X_n$. The following is a finite set of constant definitions.

$$\begin{array}{rcl} C_1 & = & E_1[C_1/X_1, \ldots, C_n/X_n], \\ & \vdots & \\ C_n & = & E_n[C_1/X_1, \ldots, C_n/X_n]. \end{array}$$

We remark that we do not allow recursively defined definitions using an infinite set of equations. A prominent feature of the constant mechanism is that it disallows the α-conversion. Take for instance the constants D_1, D_2 defined below.

$$\begin{aligned} D_1 &= \overline{a} \,|\, (a)(b.\overline{c} \,|\, D_2), \\ D_2 &= \overline{b} \,|\, (b)(a.\overline{c} \,|\, D_1). \end{aligned}$$

The global name a in the definition of D_2 is captured by the localization operator (a) in the definition of D_1 dynamically. One has that

$$\begin{aligned} D_1 &\equiv \overline{c} \,|\, (c)(b.\overline{c} \,|\, \overline{b} \,|\, (b)(c.\overline{c} \,|\, D_1)) \\ &\xrightarrow{\tau} \overline{c} \,|\, (c)(\overline{c} \,|\, (b)(c.\overline{c} \,|\, D_1)) \\ &\equiv \overline{c} \,|\, (c)(\overline{c} \,|\, (b)(c.\overline{c} \,|\, \overline{c} \,|\, (c)(b.\overline{c} \,|\, D_2))) \\ &\xrightarrow{\tau} \overline{c} \,|\, (c)(\overline{c} \,|\, (b)(\overline{c} \,|\, (c)(b.\overline{c} \,|\, D_2))). \end{aligned}$$

These internal actions are possible only if renaming never happens.

A similar way of introducing the infinite behaviors is by using the *dynamic fixpoint*, whose semantics is defined by the following rule.

$$\frac{E[\mu X.E/X] \xrightarrow{\lambda} E'}{\mu X.E \xrightarrow{\lambda} E'}$$

For the dynamic binding μ-operator, global names might get localized dynamically while unfolding the recursion. For instance $\mu X.c \,|\, (c)(\overline{c} \,|\, X)$ may not do any action if the static binding is adopted. It is divergent if the dynamic binding is admitted. For the dynamic fixpoint operations the α-conversion is also banned.

Let CCS^{def} be the CCS with the constant definition and $\text{CCS}^{d\mu}$ be the CCS with the dynamic μ-operator. These two variants of CCS are completely the same.

Fact 2 (Giambiagi, Schneider, Valencia [27])**.** $\text{CCS}^{d\mu} \sqsubseteq^{ccs}_{\downarrow} \text{CCS}^{\text{def}} \sqsubseteq^{ccs}_{\downarrow} \text{CCS}^{d\mu}$.

Proof. The conversions between $\text{CCS}^{d\mu}$ and CCS^{def} are relevant to this paper. So we take some time to explain them. Given a process P in $\text{CCS}^{d\mu}$, we may convert it into a finite set of constant definitions in the following manner.

1. To start with we introduce the constant $C = P$.
2. Suppose the conversion has introduced the following set of constant definitions.

$$\begin{aligned} C &= E_0, \\ C_1 &= E_1, \\ &\vdots \\ C_n &= E_n. \end{aligned}$$

If $\mu X.E_{n+1}$ is a subexpression in one of $E_1, \ldots, E_n$ such that E_{n+1} does not contain any occurrence of the μ-operator, then we replace the above set of definitions by the following set of definitions.

$$\begin{aligned} C &= E_0[C_{n+1}/\mu X.E_{n+1}], \\ C_1 &= E_1[C_{n+1}/\mu X.E_{n+1}], \\ &\vdots \\ C_n &= E_n[C_{n+1}/\mu X.E_{n+1}], \\ C_{n+1} &= E_{n+1}[C_{n+1}/X]. \end{aligned}$$

3. The second step is repeated until there is no occurrence of μ-operator in any of the definitions.

It is easy to see that P is structurally bisimilar to C.

Next we explain the converse conversion. Without loss of generality, suppose that we have in $\mathrm{CCS}^{\mathrm{def}}$ the following set of constant definitions.

$$\begin{aligned} C_1 &= E_1[C_1/X_1, C_2/X_2], \\ C_2 &= E_2[C_1/X_1, C_2/X_2]. \end{aligned}$$

Now C_1, C_2 form a solution to the following equations.

$$\begin{aligned} X_1 &= E_1, \\ X_2 &= E_2. \end{aligned}$$

Using the dynamic μ-operator of $\mathrm{CCS}^{d\mu}$, one could construct the process expression $\mu X_1.E_1$. Substituting $\mu X_1.E_1$ for X_1 in the second equation gives rise to the equation

$$X_2 = E_2[\mu X_1.E_1/X_1]$$

from which we could construct $\mu X_2.E_2[\mu X_1.E_1/X_1]$. It is easy to see that

$$\begin{aligned} C_1 &\sim \mu X_1.E_1[\mu X_2.E_2[\mu X_1.E_1/X_1]/X_2], \\ C_2 &\sim \mu X_2.E_2[\mu X_1.E_1/X_1]. \end{aligned}$$

We are done. □

From now on we shall ignore $\mathrm{CCS}^{d\mu}$. For the purpose of bookkeeping we mention that there is another way of introducing the infinite behaviors using *parametric definition*. Let $\mathrm{CCS}^{\mathrm{pdef}}$ be CCS with parametric definition. It is proved in [27] that $\mathrm{CCS}^{\mathrm{pdef}} \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{\mathrm{def}} \sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{\mathrm{pdef}}$. We will formally introduce the parametric definition in Section 3.

Putting the issue of expressiveness aside for a moment, constant definition does allow us to define interesting processes. Consider the process Counter defined in CCS$^{\mathrm{def}}$.

$$\begin{array}{rcl} Counter & = & zero.Counter + inc.(d)(O \,|\, d.Counter), \\ O & = & dec.\overline{d} + inc.(e)(E \,|\, e.O), \\ E & = & dec.\overline{e} + inc.(d)(O \,|\, d.E). \end{array}$$

It is a simplified version of the counter defined in [7]. A process may increment or decrement *Counter* by interacting with it at the interfaces *inc* and *dec* respectively. The process may also test if the current value of the counter is zero through interaction at *zero*. Obviously *Counter* has the following operational behavior

$$Counter \underbrace{\xrightarrow{inc} \ldots \xrightarrow{inc}}_{i \text{ times}}\underbrace{\xrightarrow{dec} \ldots \xrightarrow{dec}}_{i \text{ times}} \xrightarrow{zero} \tag{21}$$

for every natural number i. The process *Counter* is capable of remembering the difference between the number of *inc*'s it has performed and the number of *dec*'s it has done. After it has done consecutive *inc*'s for a number of times, it must do precisely the same number of *dec*'s before it can do the action *zero*. Counter is about the ability to remember the past in the presence of the fact that the potential configurations of the history are infinite. It requires the model to have some kind of recording mechanism. For a process calculus, the ability to define a counter says a great deal about Turing computability of the calculus.

2.5.1 Fixpoint, or Constant Definition

The *Counter* defined in the previous subsection is interesting in that it rules out the possibility for CCS$^{\mathrm{def}}$ to hold of such property as stated in Proposition 4. But how about CCS? It appears that CCS is closer to CCS^m than to CCS$^{\mathrm{def}}$. We now justify this intuition.

A simple idea to convert a CCS process to a CCS^m process is to rewrite every binary choice, say $A+B$, to a mixed choice, say $a.A+a.B$. For the conversion to cause no confusion, it is required that a does not appear in $A \,|\, B$. Formally we define name indexed maps from CCS to CCS^m by structural induction.

$$\begin{array}{rcl} \mathcal{I}_a(\mathbf{0}) & \stackrel{\mathrm{def}}{=} & \mathbf{0}, \\ \mathcal{I}_a(Y) & \stackrel{\mathrm{def}}{=} & Y, \\ \mathcal{I}_a(\lambda.E) & \stackrel{\mathrm{def}}{=} & \lambda.\mathcal{I}_a(E), \\ \mathcal{I}_a(E_1 \,|\, E_2) & \stackrel{\mathrm{def}}{=} & \mathcal{I}_a(E_1) \,|\, \mathcal{I}_a(E_2), \\ \mathcal{I}_a(E_1+E_2) & \stackrel{\mathrm{def}}{=} & a.\mathcal{I}_a(E_1) + a.\mathcal{I}_a(E_2), \\ \mathcal{I}_a((c)E) & \stackrel{\mathrm{def}}{=} & (c)\mathcal{I}_a(E), \\ \mathcal{I}_a(\mu Z.E) & \stackrel{\mathrm{def}}{=} & \mu Z.\mathcal{I}_a(E). \end{array}$$

The properties of $\mathcal{I}_a$ are stated in the following three lemmas.

Lemma 40. *If $P \xrightarrow{\lambda} Q$ and a does not appear in $P \,|\, Q$ then there exists some $i \geq 0$ such that $\mathcal{I}_a(P)\underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{i \text{ times}} \xrightarrow{\lambda} \mathcal{I}_a(Q)$.*

Proof. By using the equivalence $\mathcal{I}_a(E\{F/Z\}) \equiv \mathcal{I}_a(E)\{\mathcal{I}_a(F)/Z\}$, a simple induction on derivation suffices. $\square$

Using simple induction, we could also prove the converse of Lemma 40.

Lemma 41. *If $\mathcal{I}_a(P)\underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{i \text{ times}} \xrightarrow{\lambda} A$ for some natural number i and some $\lambda \neq a$, then $P \xrightarrow{\lambda} Q$ and $\mathcal{I}_a(P) \underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{j \text{ times}} \xrightarrow{\lambda} \mathcal{I}_a(Q) \underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{i-j \text{ times}} A$ for some Q and some $j \leq i$.*

Given a CCS process P, it is possible for $\mathcal{I}_a(P)$ to do an infinite sequence of a-actions. But if $\mathcal{I}_a(P)$ can do a non-a-action λ, then the number of a-actions that have to be done before λ can be enacted is bounded by a number $\mathbb{N}_a(P)$ derived from the syntax of P. We explain below how the number $\mathbb{N}_a(P)$ is calculated. Firstly we define the function $\mathbb{N}_a(\_)$ inductively on the set of all CCS$^!$ processes whose binary choices are all of the form $a.\_+ a.\_$.

$$\begin{aligned}
\mathbb{N}_a(\mathbf{0}) &\stackrel{\text{def}}{=} 1, \\
\mathbb{N}_a(\lambda.P') &\stackrel{\text{def}}{=} 1, \\
\mathbb{N}_a(P_1 \,|\, P_2) &\stackrel{\text{def}}{=} \mathbb{N}_a(P_1) + \mathbb{N}_a(P_2), \\
\mathbb{N}_a(a.P_1 + a.P_2) &\stackrel{\text{def}}{=} 1 + max\{\mathbb{N}_a(P_1), \mathbb{N}_a(P_2)\}, \\
\mathbb{N}_a((c)P') &\stackrel{\text{def}}{=} \mathbb{N}_a(P'), \\
\mathbb{N}_a(!P') &\stackrel{\text{def}}{=} 2\mathbb{N}_a(P').
\end{aligned}$$

The last line of the induction takes into consideration that an action of $!P$ is either caused by P or is caused by an interaction between two copies of P. Now for a CCS process P and a fresh name a, we define $\mathbb{N}_a(P)$ as follows:

$$\mathbb{N}_a(P) \stackrel{\text{def}}{=} \mathbb{N}_a(\mathcal{I}'(\mathcal{I}_a(P)))$$

where the translation $\mathcal{I}'$ removes the unguarded occurrence of the bound variables using the techniques developed in Section 2.3.2. The point of $\mathbb{N}_a(P)$ is that it provides an upper bound for the number of a-actions $\mathcal{I}_a(P)$ has to do before making a non-a-action.

Lemma 42. *Suppose P is a process in* CCS. *Then P cannot do any action if and only if every action sequence of $\mathcal{I}_a(P)$ with length no more than $\mathbb{N}_a(P)$ consists of only a-actions.*

Proof. Since all choices in $\mathcal{I}_a(P)$ are of the form $a.\_ + a.\_$, the translation $\mathcal{I}^{'}$ does not change the choice constructions appeared in $\mathcal{I}_a(P)$. The proofs in Section 2.3.2 establish that $\mathcal{I}^{'}(\mathcal{I}_a(P))$ is structurally bisimilar to $\mathcal{I}_a(P)$. So the minimal number of a-actions $\mathcal{I}^{'}(\mathcal{I}_a(P))$ has to do in order to do a non-a-action is the same as the minimal number of a-actions $\mathcal{I}_a(P)$ has to do in order to do a non-a-action. The lemma then follows from the definition of $\mathbb{N}_a(\_)$. □

The above lemma suggests to define the expansion order $\preceq^{ccs}$ of CCS processes in terms of the expansion order $\preceq$ in the following manner:

> For processes P, Q of CCS, $P \preceq^{ccs} Q$ if and only if $\mathcal{I}^!(\mathcal{I}_a(P)) \preceq \mathcal{I}^!(\mathcal{I}_a(Q))$ for some fresh a.

Expansion Property is a straightforward consequence of this definition.

Proposition 8. *Suppose P is a process in* CCS. *If $P_0, P_1, P_2, \ldots$ is an infinite sequence in $(\mathfrak{Drv}(P), \preceq^{ccs})$, then there is an infinite increasing subsequence $P_{i_0} \preceq^{ccs} P_{i_1} \preceq^{ccs} P_{i_2} \preceq^{ccs} \ldots$, where $i_0 < i_1 < i_2 < \ldots$.*

Compatibility Property is also valid for $\preceq^{ccs}$.

Lemma 43. *Suppose P, Q are processes in* CCS. *If $P \preceq^{ccs} Q$ and $P \xrightarrow{\lambda} P'$ then Q' exists such that $Q \xrightarrow{\lambda} Q'$ and $P' \preceq^{ccs} Q'$.*

Proof. Suppose $P \preceq^{ccs} Q$ and $P \xrightarrow{\lambda} P'$. By Lemma 40 some i exists such that $\mathcal{I}_a(P)\underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{i \text{ times}} \xrightarrow{\lambda} \mathcal{I}_a(P')$. So $\mathcal{I}^!(\mathcal{I}_a(P))\underbrace{\overset{a}{\Longrightarrow} \ldots \overset{a}{\Longrightarrow}}_{i \text{ times}} \xrightarrow{\lambda} \mathcal{I}^!(\mathcal{I}_a(P'))$. But then some Q' exists such that $\mathcal{I}^!(\mathcal{I}_a(Q))\underbrace{\overset{a}{\Longrightarrow} \ldots \overset{a}{\Longrightarrow}}_{i \text{ times}} \xrightarrow{\lambda} \mathcal{I}^!(\mathcal{I}_a(Q'))$ and $\mathcal{I}^!(\mathcal{I}_a(P')) \preceq \mathcal{I}^!(\mathcal{I}_a(Q'))$. Therefore

$$\mathcal{I}_a(Q)\underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{i \text{ times}} \xrightarrow{\lambda} \mathcal{I}_a(Q'). \tag{22}$$

It follows from (22) and Lemma 41 that $Q \xrightarrow{\lambda} Q'$. Hence $P \preceq^{ccs} Q$. □

Let $\text{CCS}^{-\text{def}}$ be the variant of CCS^- with the dynamic fixpoint operator. Similarly we have $\text{CCS}^{m\text{def}}$ and $\text{CCS}^{s\text{def}}$. What is the relationship between CCS^m and $\text{CCS}^{m\text{def}}$? Using the α-conversion one may rename the local names of a process expression in a variant with static fixpoint operations such that all names are distinct. Clearly for such a process expression dynamic and static fixpoint operations make no difference. Hence

$$\begin{aligned} \text{CCS}^- &\sqsubseteq^{ccs}_{\downarrow} \text{CCS}^{-\text{def}}, \\ \text{CCS}^s &\sqsubseteq^{ccs}_{\downarrow} \text{CCS}^{s\text{def}}, \\ \text{CCS}^m &\sqsubseteq^{ccs}_{\downarrow} \text{CCS}^{m\text{def}}. \end{aligned}$$

Busi, Gabbrielli and Zavattaro have shown in [7, 8] that the termination property is decidable in CCS^m but undecidable in CCS$^{m\text{def}}$. The technique used in their proofs essentially imply that dynamic recursion is strictly stronger interactively than static recursion in the setting of CCS. A different argument for this fact is given below.

Proposition 9. *The following properties hold:*
(i) CCS$^{-\text{def}}$ $\not\sqsubseteq^{ccs}$ CCS$^-$*;*
(ii) CCS$^{s\text{def}}$ $\not\sqsubseteq^{ccs}$ CCS^s*;*
(iii) CCS$^{m\text{def}}$ $\not\sqsubseteq^{ccs}$ CCS^m*;*
(iv) CCS$^{\text{def}}$ $\not\sqsubseteq^{ccs}$ CCS.

Proof. (ii,iii) *Counter* is actually defined in CCS$^{s\text{def}}$. Now suppose there were a process D of CCS^m that bisimulated *Counter* and that the simulated sequence of actions is the following:

$$D \overset{inc}{\Longrightarrow} D_1 \overset{inc}{\Longrightarrow} \ldots \overset{inc}{\Longrightarrow} D_i \overset{inc}{\Longrightarrow} D_{i+1} \ldots.$$

Since the above sequence is in CCS^m, we may apply Lemma 39 and Proposition 6 to conclude that there must exist D_p and D_q, where $p < q$, such that D_q can do whatever D_p can do. But then the following sequence of actions would be possible

$$D\underbrace{\overset{inc}{\Longrightarrow} \ldots \overset{inc}{\Longrightarrow}}_{q \text{ times}}\underbrace{\overset{dec}{\Longrightarrow} \ldots \overset{dec}{\Longrightarrow}}_{p \text{ times}} \overset{zero}{\Longrightarrow}$$

which contradicts the assumption that D bisimulated *Counter*.

(iv) In view of Lemma 43 and Proposition 8, the above argument can be repeated to show that CCS$^{\text{def}}$ $\not\sqsubseteq^{ccs}$ CCS.

(i) In CCS$^{-\text{def}}$, and CCS$^-$ as well, one could define the *internal choice* $E \dotplus F$ as follows:

$$E \dotplus F \quad \overset{\text{def}}{=} \quad (h)(h.E \,|\, h.F \,|\, \overline{h})$$

where h does not appear in $E \,|\, F$. Using internal choice one could define a weak form of the counter in CCS$^{-\text{def}}$ in the following manner.

$$\begin{array}{rcl} D' &=& zero.D' \dotplus inc.(d)(O' \,|\, d.D'), \\ O' &=& dec.\overline{d} \dotplus inc.(e)(E' \,|\, e.O'), \\ E' &=& dec.\overline{e} \dotplus inc.(d)(O' \,|\, d.E'). \end{array}$$

Although D' does not always do what the environments anticipate, it does exhibit the same memory capacity as *Counter*. For this reason the approach used above can be applied to establish (i). □

The fact that dynamic fixpoint operations, or constant definitions, are much stronger than static fixpoint operations, or replications, is reinforced by another fact: CCS with dynamic fixpoint operations and mixed choices is strong enough to code up CCS with arbitrary choices. It is a phenomenal result that brings out the power of constant definition.

$$
\begin{array}{rcl}
C^{\varphi}_{\mathbf{0}} & = & \overline{e_{\varphi}}, \\
C^{\varphi}_{\lambda.E} & = & \overline{e_{\varphi}} + \lambda.(e_{\varphi})C^{\varphi}_{E}, \\
C^{\varphi}_{(a)E} & = & (a)C^{\varphi}_{E}, \\
C^{\varphi}_{E_1+E_2} & = & \overline{e_{\varphi}} + \tau.(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}) + \tau.(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_2} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}), \\
C^{\varphi}_{E_1 \mid E_2} & = & \overline{e_{\varphi}} + \tau.(h)(V^{\varphi}_{E_1|E_2} \mid W^{\varphi}_{E_1|E_2}), \\
V^{\varphi}_{E_1|E_2} & = & \tau.(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.V^{\varphi}_{E_1|E_2}) + h.C^{\varphi}_{E_1 \mid E_2}, \\
W^{\varphi}_{E_1|E_2} & = & \tau.(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_2} \mid e_{\overline{\varphi}}.W^{\varphi}_{E_1|E_2}) + \overline{h},
\end{array}
$$

where $\varphi \in \{\top, \bot\}$, $\overline{\varphi} = \neg\varphi$ and $e_{\top}, e_{\bot}$ are fresh names.

Figure 2: Transformation from $\mathrm{CCS}^{\mathrm{def}}$ to $\mathrm{CCS}^{m\mathrm{def}}$.

Proposition 10. $\mathrm{CCS}^{\mathrm{def}} \sqsubseteq^{ccs} \mathrm{CCS}^{m\mathrm{def}} \sqsubseteq^{ccs} \mathrm{CCS}^{s\mathrm{def}}$.

The proof of $\mathrm{CCS}^{m\mathrm{def}} \sqsubseteq^{ccs} \mathrm{CCS}^{s\mathrm{def}}$ can be readily copied from the proof of Proposition 1. The proof of $\mathrm{CCS}^{\mathrm{def}} \sqsubseteq^{ccs} \mathrm{CCS}^{m\mathrm{def}}$ is complex. One needs to construct a translation from $\mathrm{CCS}^{\mathrm{def}}$ processes to $\mathrm{CCS}^{m\mathrm{def}}$ processes. To define the construction we start with a sound translation from *finite* $\mathrm{CCS}^{\mathrm{def}}$ processes to $\mathrm{CCS}^{m\mathrm{def}}$ processes. The encoding $[\![P]\!]_f$ of a finite CCS process P can be defined as follows:

$$[\![P]\!]_f \quad \stackrel{\mathrm{def}}{=} \quad \mu Z.[\![P]\!]_Z$$

where $[\![\_]\!]_Z$ is defined inductively by the following clauses:

$$
\begin{array}{rcl}
[\![\mathbf{0}]\!]_Z & \stackrel{\mathrm{def}}{=} & \tau.Z, \\
[\![\lambda.P]\!]_Z & \stackrel{\mathrm{def}}{=} & \tau.Z + \lambda.[\![P]\!]_f, \\
[\![(a)P]\!]_Z & \stackrel{\mathrm{def}}{=} & (a)[\![P]\!]_Z, \\
[\![P_1{+}P_2]\!]_Z & \stackrel{\mathrm{def}}{=} & \tau.[\![P_1]\!]_Z + \tau.[\![P_2]\!]_Z, \\
[\![P_1 \mid P_2]\!]_Z & \stackrel{\mathrm{def}}{=} & (h)(\mu Z'.(\tau.[\![P_1]\!]_{Z'}{+}h.Z) \mid \mu Z'.(\tau.[\![P_2]\!]_{Z'}{+}\overline{h})).
\end{array}
$$

The intuition about $[\![P]\!]_Z$ is that it retains all the capabilities of P using only separated choices. To accomplish that, $[\![P]\!]_Z$ introduces a number of computational choices in such a way that all these internal choices can be undone by going back to where it started. In $[\![P]\!]_Z$ there are a number of loops or, as we prefer to call them, *internal loopings*. Here Z is the starting point to which an internal looping may go to and by doing so it forgets all the choices it has made. The loops may be connected so that an internal looping cannot only go back locally but also globally. The soundness of the encoding $[\![\_]\!]_f$ is guaranteed by Computation Lemma.

Unfortunately the above translation cannot be extended to deal with the infinite behaviors. Suppose we were to define $[\![\mu X.E_1]\!]_f$. A straightforward approach is to let it be $[\![E_1\{\mu X.E_1/X\}]\!]_f$. By structural property, each occurrence of $\mu X.E_1$ would get interpreted as $[\![\mu X.E_1]\!]_Z$ for some bound Z. We were in a non-well-founded situation since the interpretation $[\![\mu X.E_1]\!]_Z$ would depend on some $[\![\mu X.E_1]\!]_{Z'}$ for some other Z'.

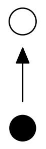
One way to get out of the awkward cycle is to use the first order pointers, some special names, to access the backup points. These pointers must be carefully introduced such that only a finite number of names are necessary. Dynamic binding plays a crucial role here.

We define a translation from CCS$^{\text{def}}$ to CCS$^{m\text{def}}$ in four steps.

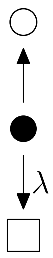
1. First a translation from CCS$^{\text{def}}$ expressions without any μ-operations to constant definitions in CCS$^{m\text{def}}$ is defined. This translation, given in Figure 2, is a first order counterpart of the above encoding. For each CCS$^{\text{def}}$ expression E without μ-operator, two constants $C_E^\top, C_E^\bot$ are introduced, one of which is indexed by the boolean value truth $\top$ and the other by false $\bot$. These two translations are defined uniformly using a boolean variable φ. The names $e_\top$ and $e_\bot$ are used to trace back to the starting point of an internal looping. The alternating use of $e_\top, e_\bot$ is essential, and is reminiscent of the alternating use of d, e in the counter defined in Section 2.5. If E contains free process variables $Y_1, \ldots, Y_m$, then $C_E^\top$ and $C_E^\bot$ are parameterized over the constants $C_{Y_1}^\top, \ldots, C_{Y_m}^\top, C_{Y_1}^\bot, \ldots, C_{Y_m}^\bot$.

   The translation $C_E^\top$, or $C_E^\bot$, constructs a structure that adds internal tau actions on the top of the structures of E. These internal actions are introduced to force all choices to be separated choices. The additional choices by the extra tau actions must be done in such a way that they can be undone if wanted. Let's explain the idea using diagrams.

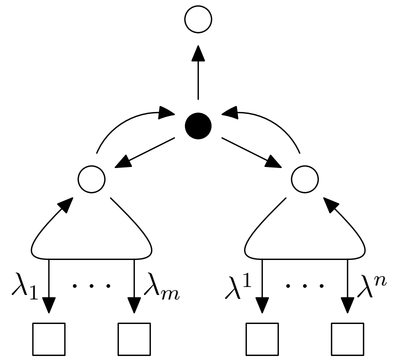
   - $C_{\mathbf{0}}^\varphi$ is able to go back to the previous checkpoint by synchronizing through name e_φ. The name e_φ acts as a pointer to the backup at the previous checkpoint. In the following diagram, the bullet represents the current state and the circle indicates the checkpoint.



   - $C_{\lambda.E}^\varphi$ has to make a decision. Either it chooses to call up the backup at the previous checkpoint by synchronizing through name e_φ, or it does the λ-action prescribed by $\lambda.E$ and forgets all about the history. In $(e_\varphi)C_E^\varphi$ the local name e_φ sets up a new checkpoint. In the following diagram the box represents the state after the λ-action.

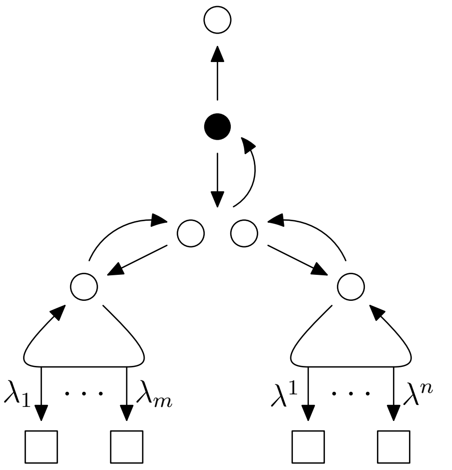


- The intended meaning of the constant $C^{\varphi}_{(a)E}$ should be clear.
- The constant $C^{\varphi}_{E_1+E_2}$ can make a decision among three choices. It may call up the backup at the previous checkpoint. It can do an internal τ-action, part of the internal looping, and then turns into $(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2})$. The process expression $(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2})$ consists of two parts. The expression $e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}$ records the backup $C^{\varphi}_{E_1+E_2}$ at the present checkpoint. When $C^{\overline{\varphi}}_{E_1}$ is engaged in an internal looping, it retains the right to reset the system to the previous state $C^{\varphi}_{E_1+E_2}$ by synchronizing at $e_{\overline{\varphi}}$. But once $C^{\overline{\varphi}}_{E_1}$ has done an action prescribed by E_1, the alternating use of e_{φ} and $e_{\overline{\varphi}}$ makes sure that the checkpoint and the backup $e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}$ are completely forgotten. It should be pointed out that in the internal looping states of $(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2})$, the system still reserves the right to kick off any actions of E_2. The third choice is symmetric to the second one.



- The constant $C^{\varphi}_{E_1 \mid E_2}$ is a little more complex than $C^{\varphi}_{E_1+E_2}$. Again it may go back to the previous state by synchronizing at e_{φ}. It may also evolve to $(h)(V^{\varphi}_{E_1|E_2} \mid W^{\varphi}_{E_1|E_2})$. Now $V^{\varphi}_{E_1|E_2}$ can evolve to $(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.V^{\varphi}_{E_1|E_2})$. While $C^{\overline{\varphi}}_{E_1}$ is looping, it can trigger $e_{\overline{\varphi}}.V^{\varphi}_{E_1|E_2}$ so that the system goes back to $V^{\varphi}_{E_1|E_2}$. If $C^{\overline{\varphi}}_{E_1}$ chooses to do an action defined by E_1, the checkpoint and the backup $e_{\overline{\varphi}}.V^{\varphi}_{E_1|E_2}$ are thrown away. The system $W^{\varphi}_{E_1|E_2}$ plays the similar role. It does for E_2 what $V^{\varphi}_{E_1|E_2}$ does for E_1. Finally when both $V^{\varphi}_{E_1|E_2}$ and $W^{\varphi}_{E_1|E_2}$ are evolving independently, they may agree to go back to $C^{\varphi}_{E_1 \mid E_2}$

together. This is achieved by a synchronization at h. The ability to synchronize at h is indicated in the diagram by two neighboring cycles and the arrow pointing to the bullet.



2. Secondly each CCS$^{\text{def}}$ process P with n occurrences of the μ-operator is unrolled to a set of n equations

$$\begin{aligned} X_1 &= E_P^1, \\ &\vdots \\ X_n &= E_P^n, \end{aligned}$$

in the way as explained in the proof of Fact 2. Let E_P be the CCS$^{\text{def}}$ expression obtained by replacing the n fixpoints in P by the corresponding process variables $X_1, \dots, X_n$. We then have a new equation

$$X = E_P$$

for a fresh process variable X.

3. Thirdly we have a set of constant definitions in CCS^{dm}. This set consists of the following definitions

$$\begin{aligned} C_X^\top &= C_{E_P}^\top, \\ C_{X_1}^\top &= C_{E_P^1}^\top, \\ &\vdots \\ C_{X_n}^\top &= C_{E_P^n}^\top, \\ C_X^\perp &= C_{E_P}^\perp, \\ C_{X_1}^\perp &= C_{E_P^1}^\perp, \\ &\vdots \\ C_{X_n}^\perp &= C_{E_P^n}^\perp, \end{aligned}$$

plus the constant definitions introduced by $C^{\top}_{E^1_P}, \ldots, C^{\top}_{E^n_P}, C^{\perp}_{E^1_P}, \ldots, C^{\perp}_{E^n_P}$.

It should be pointed out that the encoding of a particular occurrence of the process variable X_i, say, must be either $C^{\top}_{X_i}$ or $C^{\perp}_{X_i}$. This is guaranteed by the alternating use of $e_{\top}$ and $e_{\perp}$.

4. Finally the encoding $[\![P]\!]$ is given by the following definition:

$$[\![P]\!] \quad \overset{\mathrm{def}}{=} \quad (e_{\top})C^{\top}_X.$$

The encoding $[\![\_]\!]$ so defined makes full use of dynamic binding mechanism. It is very much like *Counter* defined in Section 2.5.

To understand the encoding, the reader is advised to work out the encoding of the process $P \equiv \mu X.(a{+}b \,|\, X)$. The equations derived from this process are

$$\begin{aligned} X &= Y, \\ Y &= c + d \,|\, Y, \end{aligned}$$

which can be simplified to a single equation

$$X \quad = \quad c + d \,|\, X. \tag{23}$$

The constant definitions that correspond to (23) are given as follows:

$$\begin{aligned}
P_X &= P_{c+d\,|\,X}, \\
P_{c+d\,|\,X} &= \overline{a} + \tau.(b)(Q_c \,|\, b.P_{c+d\,|\,X}) + \tau.(b)(Q_{d\,|\,X} \,|\, b.P_{c+d\,|\,X}), \\
Q_c &= \overline{b} + c.(a)P_{\mathbf{0}}, \\
Q_{d\,|\,X} &= \overline{b} + \tau.(h)(Q^1_{d\,|\,X} \,|\, Q^2_{d\,|\,X}), \\
Q^1_{d\,|\,X} &= \tau.(a)(P_d \,|\, a.Q^1_{d\,|\,X}) + h.Q_{d\,|\,X}, \\
Q^2_{d\,|\,X} &= \tau.(a)(P_X \,|\, a.Q^2_{d\,|\,X}) + \overline{h}, \\
P_d &= \overline{a} + d.(a)P_{\mathbf{0}}, \\
P_{\mathbf{0}} &= \overline{a},
\end{aligned}$$

where we write a for $e_{\top}$, b for $e_{\perp}$, P_E for $C^{\top}_E$, and Q_E for $C^{\perp}_E$. The following internal action sequence is an instance of an internal looping of the interpretation $[\![\mu X.(c{+}d \,|\, X)]\!]$.

$$\begin{aligned}
(a)P_X &\xrightarrow{\tau} (a)(b)(Q_{d\,|\,X} \,|\, b.P_{c+d\,|\,X}) \\
&\xrightarrow{\tau} (a)(b)((h)(Q^1_{d\,|\,X} \,|\, Q^2_{d\,|\,X}) \,|\, b.P_{c+d\,|\,X}) \\
&\xrightarrow{\tau} (a)(b)((h)(Q^1_{d\,|\,X} \,|\, (a)(P_X \,|\, a.Q^2_{d\,|\,X})) \,|\, b.P_{c+d\,|\,X}) \\
&\xrightarrow{\tau} (a)(b)((h)(Q^1_{d\,|\,X} \,|\, (a)((b)(Q_c \,|\, b.P_{c+d\,|\,X}) \,|\, a.Q^2_{d\,|\,X})) \,|\, b.P_{c+d\,|\,X}) \\
&\xrightarrow{c} (a)(b)((h)(Q^1_{d\,|\,X} \,|\, (a)((b)((a)P_{\mathbf{0}} \,|\, b.P_{c+d\,|\,X}) \,|\, a.Q^2_{d\,|\,X})) \,|\, b.P_{c+d\,|\,X}.
\end{aligned}$$

The last process can be equated to d as is shown below.

$$\begin{array}{cl} & (a)(b)((h)(Q^1_{d\,|\,X}\,|\,(a)((b)((a)P_{\mathbf{0}}\,|\,b.P_{c+d\,|\,X})\,|\,a.Q^2_{d\,|\,X}))\,|\,b.P_{c+d\,|\,X} \\ \sim & (a)(b)((h)(Q^1_{d\,|\,X}\,|\,(a)((b)(\mathbf{0}\,|\,b.P_{c+d\,|\,X})\,|\,a.Q^2_{d\,|\,X}))\,|\,b.P_{c+d\,|\,X}) \\ \sim & (a)(b)((h)(Q^1_{d\,|\,X}\,|\,(a)(\mathbf{0}\,|\,a.Q^2_{d\,|\,X}))\,|\,b.P_{c+d\,|\,X}) \\ \sim & (a)(b)((h)(Q^1_{d\,|\,X}\,|\,\mathbf{0})\,|\,b.P_{c+d\,|\,X}) \\ \sim & (a)((h)(Q^1_{d\,|\,X})\,|\,\mathbf{0}) \\ \approx & d. \end{array}$$

In this example, four steps of internal looping are made before the action c is performed. In the first three steps of internal looping, the agent wanders along the summation and the composition structures of $c + d\,|\,X$ and then reaches a recurrence of P_X. After the fourth step the agent reserves the right to go back to any of the previous states. But once it kicks off the action c, it throws away all the pointers that lead to the backups. This is evidenced by the strong equivalences. The last equivalence says that the sequence of actions correspond to $\mu X.(c{+}d\,|\,X) \xrightarrow{c} d$.

Having explained the encoding in detail, we now make some further comments about some design decisions of the encoding.

- Does the encoding provide a translation from CCS to CCS^m? The answer is definitely negative. The problem is to do with infinite behaviors. In the interpretation of say (23), the constant P_X plays two roles: First it represents the interpretation of the fixpoint $\mu X.(c + d\,|\,X)$; Second it is a state that can turn the clock back. To fulfill the first role, P_X must be recursively defined. To accomplish the second task, P_X must contain a (global) name that points to the previous backup. But the interpretation P_X should not introduce additional global names apart from those already appeared in E. The only way for P_X to do both jobs is to use dynamic binding.

- Why should we stick to the two local names $e_\top, e_\bot$? Why can't we use a fresh local name every time a different local name is called for in the encoding? The reason that we cannot do that is again due to infinite behaviors. Had we done that, then the encoding of a fixpoint expression could lead to an infinite number of constant definitions.

- Can we use one local name instead of the two local names $e_\top, e_\bot$? The answer is again negative. One of the tricky part of the encoding is to define the internal looping in such a way that it can always go back to the right starting point. If $C_{E_1+E_2}$ is defined by

$$\overline{e} + \tau.(e)(C_{E_1}\,|\,e.C_{E_1+E_2}) + \tau.(e)(C_{E_2}\,|\,e.C_{E_1+E_2})$$

using only one pointer e, then after $C_{E_1+E_2}$ wanders to $(e)(C_{E_1}\,|\,e.C_{E_1+E_2})$ it can never go back to $C_{E_1+E_2}$. Another possibility would be not to use

$$\begin{tikzcd}
\mathrm{CCS}^{s\mathrm{def}} \arrow[r, shift left] & \mathrm{CCS}^{\mathrm{def}} \arrow[l, shift left] \arrow[r, shift left] \arrow[d, shift left, "/" marking] & \mathrm{CCS}^{m\mathrm{def}} \arrow[l, shift left] \\
\mathrm{CCS}^{s} \arrow[d, shift left] & \mathrm{CCS} \arrow[u, shift left] \arrow[d, shift left, "/" marking] & \mathrm{CCS}^{m} \arrow[d, shift left] \\
\mathrm{CCS}^{s!} \arrow[u, shift left] \arrow[r, shift left] & \mathrm{CCS}^{!} \arrow[l, shift left] \arrow[u, shift left] \arrow[r, shift left] & \mathrm{CCS}^{m!} \arrow[l, shift left] \arrow[u, shift left]
\end{tikzcd}$$

Figure 3: Subbisimilarities of CCS

the operator (e) locally. But then things would be messed up when trying to go back to the previous backups. So one local name is not an option at all.

In some sense the dynamic fixpoint and the names $e_\top, e_\perp$ are forced upon us.

The proof of the correctness of the encoding can be found in Appendix A.

2.6 Interaction Hierarchy of CCS

We can now summarize the main results of Section 2.

Theorem 1. *The subbisimilarity relationships between the nine variants of* CCS *are depicted in Figure 3.*

In Figure 3 an arrow $\rightarrow$ indicates the existence of a subbisimilarity and $\not\rightarrow$ means the nonexistence of such a subbisimilarity. In other words, $\mathrm{CCS}^1 \rightarrow \mathrm{CCS}^2$ means that CCS^2 is at least as expressive as CCS^1; and $\mathrm{CCS}^1 \not\rightarrow \mathrm{CCS}^2$ means that there is a process in CCS^1 that cannot be expressed in CCS^2. It is clear from Figure 3 that all the subbisimilarity relationships between the nine variants of CCS have been unveiled.

The second major result tells the story of the expressiveness of CCS when Computation Criterion is applied.

Theorem 2. *The codivergent subbisimilarity relationships between the nine variants of* CCS *are depicted in Figure 4.*

In Figure 4 the tailed arrow $\rightarrowtail$ represents a codivergent subbisimilarity and $\not\rightarrowtail$ rules out any codivergent subbisimilarity.

It should be remarked that the diagram in Figure 4 does not spell out everything. Some positive and negative results have either been proved before or can be easily derived by transitivity. In fact *all* the codivergent subbisimilarity relationships among the nine CCS variants are known. The following summarizes the facts that are not explicitly indicated in the diagram of Figure 4.

Fact 3. *The following statements are valid.*
(i) $\mathrm{CCS}^{\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}^{m}$, $\mathrm{CCS}^{m} \sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{\mathrm{def}}$.

$$\begin{tikzcd}
\mathrm{CCS}^{\mathrm{def}} \arrow[r, tail, shift right, "/" marking] \arrow[d, tail, shift left, "/" marking] & \mathrm{CCS} \arrow[l, tail, shift right] \arrow[r, tail, shift right, "/" marking] & \mathrm{CCS}^{!} \arrow[l, tail, shift right] \arrow[d, tail, shift left] \\
\mathrm{CCS}^{m\mathrm{def}} \arrow[u, tail, shift left] \arrow[r, tail, shift right, "/" marking] \arrow[d, tail, shift left, "/" marking] & \mathrm{CCS}^{m} \arrow[l, tail, shift right] \arrow[r, tail, shift right] & \mathrm{CCS}^{m!} \arrow[l, tail, shift right] \arrow[u, tail, shift left] \arrow[d, tail, shift left, "/" marking] \\
\mathrm{CCS}^{s\mathrm{def}} \arrow[u, tail, shift left] \arrow[r, tail, shift right, "/" marking] & \mathrm{CCS}^{s} \arrow[l, tail, shift right] \arrow[r, tail, shift right] & \mathrm{CCS}^{s!} \arrow[l, tail, shift right] \arrow[u, tail, shift left]
\end{tikzcd}$$

Figure 4: Codivergent Subbisimilarities of CCS

(ii) $\mathrm{CCS}^{\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}^{s}$, $\mathrm{CCS}^{s} \sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{\mathrm{def}}$.
(iii) $\mathrm{CCS}^{m\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}$, $\mathrm{CCS} \not\sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{m\mathrm{def}}$.
(iv) $\mathrm{CCS}^{m\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}^{s}$, $\mathrm{CCS}^{s} \sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{m\mathrm{def}}$.
(v) $\mathrm{CCS}^{s\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}$, $\mathrm{CCS} \not\sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{s\mathrm{def}}$.
(vi) $\mathrm{CCS}^{s\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}^{m}$, $\mathrm{CCS}^{m} \not\sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{s\mathrm{def}}$.

Proof. All the positive results are immediate by composition. The proofs of $\mathrm{CCS}^{\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}^{m}$, $\mathrm{CCS}^{\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}^{s}$, $\mathrm{CCS}^{m\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}$, $\mathrm{CCS}^{m\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}^{s}$, $\mathrm{CCS}^{s\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}$ and $\mathrm{CCS}^{s\mathrm{def}} \not\sqsubseteq^{ccs} \mathrm{CCS}^{m}$ are basically an iteration of the proof of (iv) of Proposition 9. The proof of $\mathrm{CCS}^{m} \not\sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{s\mathrm{def}}$ is the same as that of Proposition 3. In view of the fact that $\mathrm{CCS} \not\sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{s\mathrm{def}}$ follows from $\mathrm{CCS} \not\sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{m\mathrm{def}}$, the only thing left is to prove $\mathrm{CCS} \not\sqsubseteq^{ccs}_{\downarrow} \mathrm{CCS}^{m\mathrm{def}}$.

We show that *RanTimer* cannot be bisimulated by any terminating process of $\mathrm{CCS}^{m\mathrm{def}}$. The crucial property about *RanTimer* is that it has an infinite number of s-derivatives. We prove by structural induction the following fact: A process expression E of $\mathrm{CCS}^{m\mathrm{def}}$ has only a finite number of s-derivatives if it satisfies the following three conditions about its immediate actions.

1. E cannot do any τ-action;
2. E can do at least one s-action; and
3. E cannot do two consecutive s-actions.

Here is the structural induction.

1. If E is a summation $\sum_{i\in I} \lambda_i.E_i$ then clearly E has a finite number of s-derivatives.
2. Suppose E is a composition $E_0 \mid E_1$. If E_0 (or E_1) can do an immediate s-action, then E_1 (or E_0) can never ever do any s-action. By induction hypothesis E_0 (or E_1) has a finite number of s-derivatives. So E has a finite number of s-derivatives.
3. If E is $(c)E'$ then by structural induction E' has a finite number of s-derivatives. Consequently E has a finite number of s-derivatives.

4. Now suppose that $E \equiv \mu X.E'$. We show by structural induction that all the occurrences of X in E' are guarded. Assume that there were an unguarded occurrence of X in E'.

   - E' cannot be a mixed choice since all occurrences of X in a mixed choice are guarded.
   - If $E' \equiv E_0 \,|\, E_1$, then we may assume without loss of generality that the unguarded occurrence of X is in E_0. In this case E_1 does not have any s-derivatives for otherwise E would be able to do two consecutive s-actions. So we may continue the structural induction with E_0.
   - If E' is $(c)E''$ then we continue the structural induction on E''.
   - $E' \equiv \mu Z.E''$. In this case we continue the structural induction on E''.

   The above structural induction must stop. In other words, there is no unguarded occurrence of X in E'. It follows that E has only a finite number of s-derivatives.

A terminating process $\mathrm{CCS}^{m\mathrm{def}}$ must meet the above three conditions if it bisimulated *RanTimer*. Such a process does not exist by the above argument. □

Several general conclusions about the expressiveness of CCS can be drawn from the interaction hierarchy of CCS.

- Constant definition in CCS is strictly more expressive than fixpoint operator. Fixpoint in CCS is strictly more expressive than replication.
- General choice in CCS is strictly more expressive than guarded choice. If all choices are guarded, then fixpoint operator is equivalent to replication operator.
- There are specifications written in CCS that cannot be implemented by CCS processes without the choice operators. An automaton is fundamentally different from a concurrent system.

These conclusions are also of practical interest. In most applications, CCS with constant definition and guarded choice is preferred.

Theorem 1 and Theorem 2 point out that dynamic binding, admitted by constant definition and dynamic μ-operator, is strictly more powerful than static binding in CCS. The essential reason for the difference is that dynamic binding achieves a certain degree of name-passing through process relocation. We would gain more insight if the following question can be answered: What would happen to the relative expressiveness if CCS is extended with name-passing communication mechanism? We will soon know the answer.

3 Subbisimilarity for Pi

In this section we apply the idea of subbisimilarity to investigate the interactability of the variants of π-calculus. Most results we have established for CCS carry over to π-calculus. To make our transformation from CCS to π smooth, we shall be using a particular presentation of π-calculus.

We start by explaining an unusual feature of the standard presentation of π. In the pioneering paper on π-calculus [43], as well as a huge number of following works [60], π-calculus and its variants are defined by adopting a uniform set of names. A name has both a constanthood and a variablehood. The name x in $\overline{x}y.A$ is a constant name when it acts as a channel to communicate. On the other hand, the name x in $y(x).\overline{x}y.A$ has to be a variable name in a correct understanding of the semantics of π-calculus. This confusion of constant names and variable names has made the theory of π-calculus more complex than it should be.

In this paper we adopt the definition of π-calculus given in [23]. We take the view that all names are constant. In order to define mobility, we introduce *name variables.* Let $\mathcal{N}_v$ be the set of name variables. The following convention will be maintained in the rest of the paper.

- $\mathcal{N}$ will be ranged over by a, b, c, d, e, f, g, h.
- $\mathcal{N}_v$ will be ranged over by u, v, w, x, y, z.
- $\mathcal{N} \cup \mathcal{N}_v$ will be ranged over by l, m, n, o, p, q.

An *assignment* ρ is a map $\mathcal{N}_v \rightarrow \mathcal{N}$. A *substitution* σ is a partial map $\mathcal{N}_v \rightharpoonup \mathcal{N} \cup \mathcal{N}_v$ whose domain of definition is finite. A substitution is often given by the notation $\{n_1/x_1, \ldots, n_i/x_i\}$, where $\{x_1, \ldots, x_i\}$ is the domain of the substitution and $\{n_1, \ldots, n_i\}$ is the range of the substitution. Both assignment and substitution are used in postfix manner.

The syntax of the core π-calculus is given by the following grammar.

$$E \ := \ \mathbf{0} \mid n(x).E \mid \overline{n}m.E \mid \tau.E \mid E \,|\, E \mid (a)E \mid \ \mu X.E.$$

We shall let π^- denote this calculus. The set of the action labels is $\{ac, \overline{a}c, \overline{a}(c) \mid a, c \in \mathcal{N}\} \cup \{\tau\}$. There are input actions like ac, output actions like $\overline{a}c$, and restricted output actions like $\overline{a}(c)$. In $ac, \overline{a}c, \overline{a}(c)$, the name a is *subject name* and the name c is *object name.* The derived notation $\overline{a}(c).E$ stands for $(c)\overline{a}c.E$. We will abbreviate $\overline{a}(c).E$ to $\overline{a}.E$ if c does not appear in E. Similarly $a.E$ stands for some $a(x).E$ such that x does not occur in E.

The rules that introduce the operational semantics of π^- are defined below.

Prefix

$$\frac{}{a(x).E \xrightarrow{ac} E\{c/x\}} \qquad \frac{}{\overline{a}c.E \xrightarrow{\overline{a}c} E} \qquad \frac{}{\tau.E \xrightarrow{\tau} E}$$

Composition

$$\frac{E \xrightarrow{\lambda} E'}{E \,|\, F \xrightarrow{\lambda} E' \,|\, F} \qquad \frac{E \xrightarrow{ac} E' \quad F \xrightarrow{\overline{a}c} F'}{E \,|\, F \xrightarrow{\tau} E' \,|\, F'} \qquad \frac{E \xrightarrow{ac} E' \quad F \xrightarrow{\overline{a}(c)} F'}{E \,|\, F \xrightarrow{\tau} (c)(E' \,|\, F')}$$

Restriction

$$\frac{E \xrightarrow{\overline{a}c} E'}{(c)E \xrightarrow{\overline{a}(c)} E'} \qquad \frac{E \xrightarrow{\lambda} E' \quad c \text{ does not appear in } \lambda}{(c)E \xrightarrow{\lambda} (c)E'}$$

Fixpoint

$$\frac{E\{\mu X.E/X\} \xrightarrow{\lambda} E'}{\mu X.E \xrightarrow{\lambda} E'}$$

For a comprehensive account of the theory of π-calculus, one could consult [23].

The power of π-calculus comes from the ability to create potentially infinite number of new names during execution. The following example illustrates the point. Let N abbreviate the process

$$(b)(\overline{b}c_0 \,|\, \mu X.(\overline{a}a{+}(b(z).\overline{z}(c').\overline{b}c' \,|\, X))).$$

It is obvious that N admits the infinite action sequence

$$N \xrightarrow{\tau}\xrightarrow{\overline{c_0}(c_1)}\xrightarrow{\tau}\xrightarrow{\overline{c_1}(c_2)}\xrightarrow{\tau}\xrightarrow{\overline{c_2}(c_3)}\ldots.$$

It can also perform the finite action sequence

$$N \xrightarrow{\overline{a}a}\xrightarrow{\tau}\xrightarrow{\overline{c_0}(c_1)}\xrightarrow{\tau}\xrightarrow{\overline{c_1}(c_2)}\ldots\xrightarrow{\tau}\xrightarrow{\overline{c_i}(c_{i+1})}\sim \mathbf{0}$$

for all $i \geq 0$. A variant of the above process is the following

$$(b)(\overline{c_0}(c').\overline{b}c' \,|\, \mu X.(b(z).\overline{z}(c'').\overline{b}c'' \,|\, X))$$

which can bisimulate a process N' whose *only* action sequence is the following:

$$N' \xrightarrow{\overline{c_0}(c_1)}\xrightarrow{\overline{c_1}(c_2)}\xrightarrow{\overline{c_2}(c_3)}\ldots. \tag{24}$$

We shall see that (24), without any tau actions between any observable actions, can be achieved by parametric definition.

Different choice operators induce different variants of π^-:

- π is π^- extended with binary choice;
- π^m is π^- extended with mixed choice;
- π^s is π^- extended with separated choice.

By replacing fixpoint operator by replicator, one gets $\pi^!$, $\pi^{m!}$ and $\pi^{s!}$.

The interactability of these variants of π-calculus depends to a great extent on their infinite behaviors. So the question comes naturally, what operational properties do π-variants inherit from CCS variants? An important aspect of the operational semantics is described in Lemma 36. Does $\pi^!$ for instance enjoy the expansion property? *Ex post facto*, we could say that the motivation of

π-calculus is precisely to defeat Lemma 36. The input prefixes of π draw input values from an infinite set of names. So we can have following infinite behavior:

$$(b)(c_0(z).\overline{b}z \,|\, !b(y).y(z).\overline{b}z) \xrightarrow{c_0c_1}\xrightarrow{\tau}\xrightarrow{c_1c_2}\xrightarrow{\tau}\xrightarrow{c_2c_3}\xrightarrow{\tau}\ldots.$$

No two processes in the above sequence are comparable in terms of expansion order. Similarly the localization operator combined with the output operator have the ability to generate an infinite number of new names. Let L be the process $\overline{b}c_0.\overline{c_0}d \,|\, !b(z).\overline{z}(c').\overline{b}c'.\overline{c'}d$ in $\pi^!$. In the following infinite action sequence

$$L \xrightarrow{\tau}\xrightarrow{\overline{c_0}(c_1)}\xrightarrow{\tau}\xrightarrow{\overline{c_1}(c_2)}\xrightarrow{\tau}\xrightarrow{\overline{c_2}(c_3)}\ldots$$

no two processes are compatible with respect to a syntactic order like $\preceq$. The conclusion we draw from the examples is that we cannot expect to have a result in $\pi^!$ as general as Lemma 36. What we can discuss is which infinite action sequences are well quasi-ordered. As it turns out, there is not too much to say on this. The counter is definable in $\pi^!$ by simplifying the register given in [7]. The two concurrent components of the counter are given by the following recursive definitions.

$$\begin{aligned} Zero &\overset{\text{def}}{=} (c)(\overline{c} \,|\, !c.(zero.\overline{c} + inc.\overline{succ}(d).d.\overline{c})),\\ Succ &\overset{\text{def}}{=} !succ(x).(c)(\overline{c} \,|\, !c.(dec.\overline{x} + inc.\overline{succ}(d).d.\overline{c})). \end{aligned}$$

The process $Counter^\pi$, the counter in π, can be defined as follows.

$$Counter^\pi \overset{\text{def}}{=} (succ)(Zero \,|\, Succ).$$

It is pointed out in [7] that $Counter^\pi$ can be defined in a very small fragment of πI-calculus [54] with a limited use of the restricted output. The implication of this fact is that one should not expect a well quasi-order in the π-like calculi if localization operator plays a significant role. Having said that, we still have a general guideline for a well quasi-ordered infinite action sequence. In the following lemma $Csub(P_i)$ is the set of the concurrent subexpressions of P_i and $depth_{()}(P_i)$ is the depth of the nested localization operations, the definitions of which are the same as in CCS$^!$.

Lemma 44. *In $\pi^!$ the infinite action sequence*

$$P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} P_2 \xrightarrow{\lambda_2} \ldots$$

is well quasi-ordered by $\preceq$ if the followings are met.

1. *$\bigcup_{i\in\omega} Csub(P_i)$ is finite;*
2. *$\bigcup_{i\in\omega} ln(P_i)$ is finite;*
3. *$\mathbf{sup}_{i\in\omega}\{depth_{()}(P_i)\} < \omega$.*

The three conditions of the lemma guarantee that the proof of Lemma 36 can be used in the present setting.

For an application of the lemma, suppose

$$P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} P_2 \xrightarrow{\lambda_2} \ldots$$

is an infinite action sequence in $\pi^!$. It is well quasi-ordered if the following conditions are met.

- If P_0 contains an output prefix $\overline{a}c$ then c is not restricted by a localization operator (c) in P_0.
- The set $\{c \mid \exists i{\geq}0.\exists a{\in}\mathcal{N}.\lambda_i = ac\}$ is finite.

Since only a finite number of new names are introduced, the conditions of Lemma 44 are satisfied. There are interesting cases where there are new names produced by restricted output actions but the number of such names is finite. Since $\pi^!$ is Turing complete, there is no algorithm to check if an internal looping of a process of $\pi^!$ is well quasi-ordered.

In what follows, the random timer of π refers to the process defined by the following equation.

$$RanTimer^{\pi} \stackrel{\text{def}}{=} \mu X.(\overline{a}a + \overline{b}b \,|\, X). \tag{25}$$

3.1 Pi Subbisimilarity

To compare the interactional powers of π-variants, we need to define subbisimilarity relationship on them. Such a relationship must be strong enough to characterize the interactability of π-processes. Similar to the case of CCS, a subbisimilarity for π-variants generalizes bisimulation equivalence on one π-variant to a relationship from one π-variant to another. It is therefore instructive to take a look at the observational theory of π-calculus first.

A number of observational equivalences have been proposed and studied for π-calculus [60] formulated using a uniform treatment of names. The early equivalence [43] is defined in terms of the standard bisimulation. But to guarantee that the equivalence is closed under input prefix operation, it is required that if two processes are equivalent then all their substitution instances are equivalent as well. Consider the processes defined below.

$$\begin{aligned}
C &\stackrel{\text{def}}{=} u(x) \,|\, \overline{b}d, \\
D_0 &\stackrel{\text{def}}{=} u(x).\overline{b}d + \overline{b}d.u(x), \\
D_1 &\stackrel{\text{def}}{=} u(x).\overline{b}d + \overline{b}d.u(x) + \tau, \\
A &\stackrel{\text{def}}{=} c(u).C, \\
B_0 &\stackrel{\text{def}}{=} c(u).D_0, \\
B_1 &\stackrel{\text{def}}{=} c(u).D_1.
\end{aligned}$$

Now C is early equivalent to neither D_0 nor D_1 because A is early equivalent to neither B_0 nor B_1. However B_0+B_1 is early equivalent to $A+B_0+B_1$ since an input action of A can be simulated by either B_0 or B_1, depending on whether the input name is b or not. The action

$$A + B_0 + B_1 \xrightarrow{cz} z(x) \,|\, \overline{b}y$$

for example is simulated by

$$B_0 + B_1 \xrightarrow{cz} z(x).\overline{b}y + \overline{b}y.z(x).$$

Notice that $z(x).\overline{b}y + \overline{b}y.z(x)$ is not early equivalent to $z(x) \,|\, \overline{b}y$. In a uniform treatment of names, there are reasons that even $B_0 + B_1$ and $A + B_0 + B_1$ should not be regarded as equivalent. The argument is that in a distributed computing framework, environments are always changing; it is more reasonable to adopt a dynamic viewpoint. After $B_0 + B_1$ has simulated the cz action, the environments might have changed considerably so that they might be able to change z to b. So we need to have equivalences capable of withstanding the strong dynamic power of the environments. One such an equivalence is the open bisimilarity of Sangiorgi [55, 22] and the quasi-open bisimilarity [59, 18] of Sangirogi and Walker. Compared to the bisimilarity for CCS, the definition of the open bisimilarity is complex for the reason that one needs to deal with an infinite family of relations rather than a single relation.

Our adoption of the name dichotomy between names and name variables considerably simplifies the observational theory of π-calculus. For example the bisimulation equivalence for π-processes can be defined just like in CCS.

Definition 14. *A symmetric relation $\mathcal{R}$ on π-processes is a* π-bisimulation *if the following statement is valid.*

$$\textit{If } P\mathcal{R}Q \xrightarrow{\lambda} Q' \textit{ then } P \overset{\widehat{\lambda}}{\Longrightarrow} P'\mathcal{R}Q' \textit{ for some } P'.$$

The π-bisimilarity $\approx_\pi$ *is the largest π-bisimulation.*

If one replaces $\overset{\widehat{\lambda}}{\Longrightarrow}$ by $\xrightarrow{\lambda}$ in the above definition, one gets the *structural π-bisimilarity* $\sim$. We remark that we use the same notation $\sim$ in both π and CCS. The overloading will never cause any confusion.

There is a more contextual approach to the equational theory of π-calculus. Instead of imposing the bisimulation condition on the labeled semantics, the contextual approach relies on barbedness condition on reductional semantics. We say that a π-process P is *immediately barbed at a*, notation $P{\downarrow}a$, if $P \xrightarrow{\lambda}$ for some λ whose subject name is a, and that P is *barbed at a* if $P \Longrightarrow{\downarrow}\, a$. A binary relation $\mathcal{R}$ is *barbed* if $\forall a \in \mathcal{N}.P{\Downarrow}a \Leftrightarrow Q{\Downarrow}a$ whenever $P\mathcal{R}Q$. We are now in a position to introduce the barbed bisimilarity of Milner and Sangiorgi [45].

Definition 15. *A symmetric barbed relation $\mathcal{R}$ on π-processes is a* barbed π-bisimulation *if the following statements are valid.*

1. If $P\mathcal{R}Q \xrightarrow{\tau} Q'$ then $P \Longrightarrow P'\mathcal{R}Q'$ for some P'.

2. $\mathcal{R}$ is closed under the composition and the localization operations.

The barbed π-bisimilarity $\approx_\pi^b$ *is the largest barbed π-bisimulation.*

Using the technique of [45], it is easy to prove the following fact.

Fact 4. *The π-bisimilarity $\approx_\pi$ and the barbed π-bisimilarity $\approx_\pi^b$ coincide in every π-variant studied in this paper.*

Fact 4 is assuring in that the labeled semantics and the reductional semantics give rise to the same equivalence relation on π-processes in a bisimulation framework. It also suggests that we may use either the labeled transition approach or the reductional approach to study the relative expressiveness of π-variants. We go for the former option since it simplifies the proofs.

Definition 16. *Suppose that π^1 and π^2 are two π-variants. A* subbisimilarity *from π^1 to π^2 is a total relation $\mathcal{R} \subseteq \mathcal{P}_{\pi^1} \times \mathcal{P}_{\pi^2}$ such that the following statements are valid whenever $P\mathcal{R}Q$.*

If $P \xrightarrow{\lambda}_1 P'$ then $Q \overset{\widehat{\lambda}}{\Longrightarrow}_2 Q'$ for some Q' such that $P' \; \mathcal{R} \; Q'$.

If $Q \xrightarrow{\lambda}_2 Q'$ then $P \overset{\widehat{\lambda}}{\Longrightarrow}_1 P'$ for some P' such that $P' \; \mathcal{R} \; Q'$.

We say that π^1 is subbisimilar *to π^2, notation $\pi^1 \sqsubseteq^\pi \pi^2$, if there is a subbisimilarity from the former to the latter. The notations $\sqsubset^\pi, \sqsubseteq^\pi_\downarrow, \sqsubset^\pi_\downarrow$ are defined accordingly.*

Clearly the subbisimilarity relationship $\sqsubseteq^\pi$ is transitive. The largest subbisimilarity from a π-variant to itself is the π-bisimilarity $\approx_\pi$.

In what follows we make use of Definition 16 to classify the interactability of the name passing interactions.

3.2 Choice in Pi

The relative expressiveness of choice operators in π-calculus bears resemblance to that in CCS. As a matter of fact some of the results established in Section 2 can be readily stated for π-calculus since the proofs are almost unchanged.

Proposition 11. *$\pi^- \sqsubset^\pi \pi^m \sqsubseteq^\pi \pi^s$ and $\pi^{-\mathrm{def}} \sqsubset^\pi \pi^{m\mathrm{def}} \sqsubseteq^\pi \pi^{s\mathrm{def}}$.*

Proof. A reiteration of the proof of Proposition 1. □

If Computation Criterion must be met, the picture remains the same as in CCS.

Proposition 12. *The following statements are valid.*
(i) $\pi^s \sqsubset^\pi_\downarrow \pi^m \sqsubset^\pi_\downarrow \pi$.
(ii) $\pi^{s\mathrm{def}} \sqsubset^\pi_\downarrow \pi^{m\mathrm{def}} \sqsubset^\pi_\downarrow \pi^{\mathrm{def}}$.

Suppose P is a π-process. The encoding of P is defined as follows:

$$[\![P]\!] \stackrel{\text{def}}{=} (e)[\![P]\!]^e.$$

The indexed translation $[\![E]\!]^x$ is defined structurally as follows:

$$\begin{aligned}
[\![\mathbf{0}]\!]^p &\stackrel{\text{def}}{=} \overline{p}, \\
[\![\lambda.E]\!]^p &\stackrel{\text{def}}{=} \overline{p} + \lambda.(e)[\![E]\!]^e, \\
[\![(a)E]\!]^p &\stackrel{\text{def}}{=} (a)[\![E]\!]^p, \\
[\![E_1{+}E_2]\!]^p &\stackrel{\text{def}}{=} \mu Z.(\overline{p} + \tau.(e)([\![E_1]\!]^e \,|\, e.Z) + \tau.(e)([\![E_2]\!]^e \,|\, e.Z)), \\
[\![E_1 \,|\, E_2]\!]^p &\stackrel{\text{def}}{=} \mu Z.(\overline{p} + \tau.(d)(\mu V.(\tau.(e)([\![E_1]\!]^e \,|\, e.V) + d.Z) \\
&\qquad\qquad\qquad\qquad \,|\, \mu W.(\tau.(e)([\![E_2]\!]^e \,|\, e.W) + \overline{d}))), \\
[\![\mu X.E]\!]^p &\stackrel{\text{def}}{=} (b)(\overline{b}(c).\overline{c}p \,|\, \mu X.b(c).c(z).(\overline{p} + \tau.[\![E]\!]^z \varsigma)),
\end{aligned}$$

where $\varsigma = \{(\overline{b}(c).\overline{c}p_1 \,|\, X)/[\![X]\!]^{p_1}, \ldots, (\overline{b}(c).\overline{c}p_i \,|\, X)/[\![X]\!]^{p_i}\}$.

Figure 5: Translation from π to π^m

Proof. Both the proof of $\pi^s \sqsubset^{\pi}_{\downarrow} \pi^m$ and the proof of $\pi^m \sqsubset^{\pi}_{\downarrow} \pi$ are copies of the proof of Proposition 3. $\square$

The construction carried out in the proof of Proposition 17 indicates that some kind of dynamic binding for fixpoints can be achieved by making use of the communication mechanism of π-calculus. This property is explored in the proof of following proposition.

Proposition 13. $\pi \sqsubseteq^{\pi} \pi^m$.

Proof. We define a translation $[\![\_]\!]^p$, indexed by the names, from π-processes to π^m-processes. This is given in Figure 5. The translation is very much like the one given in Figure 2. The major differences are stated below.

- Whereas the encoding given in Figure 2 gives rise to a finite set of recursive equations, the structural definition in Figure 5 is nonrecursive. The translation $[\![P]\!]^e$ of a π-process P is a π^m-process. It is not a constant defined by equations.
- While the extra recursions introduced by the encoding $[\![\_]\!]^{e_\varphi}$ of Figure 2 are achieved through recursive constant definitions, they are realized by fixpoints in the interpretation $[\![\_]\!]^e$ of Figure 5.
- Both encodings use names to trace back to the backup points. The interpretation of Figure 2 has to use two names $e_\top, e_\bot$ in an alternating manner to keep track of the correct local pointers. On the other hand the encoding of Figure 5 does not need that dichotomy since the pointers can be locally updated by internal communications.

- The main semantic difference between the two encodings is their treatments of the recursions. In the interpretation of the constant definition

$$C = P$$

of CCS$^{\mathrm{def}}$, the interpretation of an occurrence of C that appears in P must be assigned of the correct pointer. This can only be done by using the dynamic binding of CCS$^{m\mathrm{def}}$. In the present interpretation of

$$\mu X.E$$

different occurrences of X in E get interpreted with different indexed names by α-conversion. Let $[\![X]\!]^{e_1}, \ldots, [\![X]\!]^{e_i}$ be all the distinct indexed interpretations of X in $[\![E]\!]^z$. In order for the internal computations of $[\![\mu X.E]\!]^e$ to act correctly, $[\![X]\!]^{e_1}, \ldots, [\![X]\!]^{e_i}$ must be interpreted as

$$\bar{b}(c).\bar{c}e_1 \,|\, X, \ldots, \bar{b}(c).\bar{c}e_i \,|\, X$$

respectively.

The correctness of the encoding is proved in a completely same fashion as the proof of Proposition 10. As in the proof of Proposition 10, the relation $\xrightarrow{\tau_0}$ is introduced to indicate internal computations. In

$$(b)(\bar{b}(c).\bar{c}p \,|\, \mu X.b(c).c(z).(\bar{e} + \tau.[\![E]\!]^z\varsigma))$$

which is the interpretation of $[\![\mu X.E]\!]^e$, the communications at b and c are internal computations. Let $\mathcal{R}$ be the following relation

$$\{(P, Q) \mid [\![P]\!] \xrightarrow{\tau_0}{}^{*} Q \text{ for some } \pi \text{ process } P\}.$$

The relation $\mathcal{R}$ also satisfies a kind of bisimulation property, which is a consequence of the properties described next. The first property is the following one.

If $[\![P]\!]^e \xrightarrow{\tau_0}{}^{*} Q$ then $Q \xrightarrow{\tau_0}{}^{*} Q'$ for some Q' such that $Q' \sim [\![P']\!]^e$ for some $P' \sim P$.

The proof of this property is almost the same as the proof of Lemma 51 of Appendix A. If $P \equiv \mu X.E$, then $[\![\mu X.E]\!]^e \xrightarrow{\tau_0}\xrightarrow{\tau_0}\xrightarrow{\tau_0} [\![E]\!]^e\varsigma$ are the three step internal communications forced upon $[\![\mu X.E]\!]^e$. Here ς stands for some substitution

$$\{(\bar{b}(c).\bar{c}e_1 \,|\, X)/[\![X]\!]^{e_1}, \ldots, (\bar{b}(c).\bar{c}e_i \,|\, X)/[\![X]\!]^{e_i}\}.$$

It can be easily seen that $[\![E]\!]^e\varsigma \sim [\![E\{\mu X.E/X\}]\!]^e$. Trivially $E\{\mu X.E/X\} \sim \mu X.E$. The second property is about simulation.

If $P \xrightarrow{\lambda} P'$ in π-calculus than $[\![P]\!]^e \xrightarrow{\tau_0}{}^{*}\xrightarrow{\lambda} Q'$ in π^m-calculus for some Q' such that $[\![P']\!]^e\mathcal{R} \sim Q'$.

In other words, the encoding of P simulates the operational semantics fully. The proof of the second property is similar to that of Lemma 52 of Appendix A. The third property is to do with operational reflection.

> If $[\![P]\!]^e \xrightarrow{\tau_0}{}^* \xrightarrow{\lambda} Q'$ in π^m-calculus, then $P \xrightarrow{\lambda} P'$ in π-calculus for some P' such that $P'\mathcal{R} \sim Q'$.

This last property says that the encoding simulates the operational semantics faithfully. The proof is similar to that of Lemma 53 of Appendix A.

We conclude that $\mathcal{R}$ is a subbisimilarity. □

3.3 Replication in Pi

Let's now turn to the replication operator. The proof of (ii) of Lemma 26 is done in such a way that can be routinely generalized to the π-setting. Hence the following proposition.

Proposition 14. *$\pi^m \sqsubseteq^{\pi}_{\downarrow} \pi^{m!}$ and $\pi^s \sqsubseteq^{\pi}_{\downarrow} \pi^{s!}$.*

Unlike in CCS, the converse of Proposition 14 fails. It appears that the expansion law is critical for the validity of Lemma 37.

Lemma 45. *$\pi^! \not\sqsubseteq^{\pi}_{\downarrow} \pi^{m!}$.*

Proof. Consider the process $A = f(x).(a \,|\, \overline{x} + c \,|\, \overline{x})$. Suppose there was a $\pi^{m!}$ process B that was bisimilar to A. We may assume that $B \not\twoheadrightarrow$. Now consider the action

$$f(x).(a \,|\, \overline{x} + c \,|\, \overline{x}) \xrightarrow{fb} a \,|\, \overline{b} + c \,|\, \overline{b}. \tag{26}$$

It must be simulated by

$$B \xrightarrow{fb}\Longrightarrow D \tag{27}$$

for some D such that $\neg(D \xrightarrow{\tau})$. Now D must be able to do the following actions

$$D \xrightarrow{a}, \tag{28}$$

$$D \xrightarrow{\overline{b}}, \tag{29}$$

$$D \xrightarrow{c}. \tag{30}$$

Action (28) and action (30) must be induced by a same mixed choice

$$\sum_{i\in I} \lambda_i.D_i \tag{31}$$

for otherwise D would be able to do an a-action followed by a c-action. For similar reason D must contain a mixed choice

$$\sum_{j\in J} \lambda_j.D_j \tag{32}$$

that induces (29). Choices (31) and (32) should not be distinct for otherwise D would be able to perform a $\overline{b}$-action followed by either an a-action or a c-action. Now suppose (31) and (32) are the same choice. The action sequence

$$B \xrightarrow{fa}\Longrightarrow D\{a/c\}$$

can only be matched up by

$$f(x).(a \,|\, \overline{x} + c \,|\, \overline{x}) \xrightarrow{fa} a \,|\, \overline{a} + c \,|\, \overline{a}$$

which is a contradiction since $a \,|\, \overline{a} + c \,|\, \overline{a} \xrightarrow{\tau} \mathbf{0}$ yet $D\{a/c\} \not\rightarrow$. $\Box$

An even more significant negative result about the expressiveness of $\pi^!$ is stated next.

Proposition 15. $\pi \not\sqsubseteq^{\pi}_{\downarrow} \pi^!$.

Proof. We use $RanTimer^{\pi}$, defined in (25), as the counter example. Suppose there were a process P in $\pi^!$ that bisimulated $RanTimer^{\pi}$. By codivergence property, we may assume that $P \not\rightarrow$. If we can prove that the set

$$\{P' \mid P \xrightarrow{\overline{a}a} P'\} \tag{33}$$

is finite, we may use the argument in the proof of Proposition 3 to complete the proof. The finiteness of (33) can be simply established as follows: The prefix $\overline{a}a$ that can be immediately fired cannot be underneath any replication operator. This is because $RanTimer^{\pi}$ cannot do an infinite sequence of $\overline{a}a$ action. But then a simple structural induction suffices, bearing in mind that P may not do any τ-action. $\Box$

3.4 Parametric Definition in Pi

In π-calculus dynamic binding is far more tricky than in CCS. Take for instance the process $\overline{a}c.P \,|\, (c)(R \,|\, a(z).Q)$. One way to define the operational semantics would be to admit following communication

$$\overline{a}c.P \,|\, (c)(R \,|\, a(z).Q) \xrightarrow{\tau} P \,|\, (c)(R \,|\, Q\{c/z\})$$

where the name c received through the communication gets captured dynamically. This way of defining the semantics seems to ban the α-conversion completely. But then one would have

$$(\overline{c}c \,|\, \overline{a}(c).P) \,|\, a(z).Q \xrightarrow{\tau} (c)((\overline{c}c \,|\, P) \,|\, Q\{c/z\})$$

in which $\overline{c}c$ has got bound as a side effect in a structural operational semantics! This is really something undesirable. Even with the extra help of additional syntactical gadgets to bypass this problem, there are still other unintended communications like

$$\overline{a}(c).P \,|\, a(z).(c)Q \xrightarrow{\tau} (c)(P \,|\, (c)Q\{c/z\}).$$

A standard solution to avoid such problems is to admit name escape. This makes good sense since it goes along with the idea of dynamic name binding very well. Now we can modify the above semantics in favour of the following reduction:

$$(\overline{x}x \,|\, \overline{a}(c).P) \,|\, a(z).Q \xrightarrow{\tau} (\overline{x}x \,|\, (c)P) \,|\, Q\{c/z\}.$$

The π-calculus with name escape does not have bound output actions, and according to the definition of this paper it is not a π-variant. Moreover it is considerably weaker than π-calculus proper, as we shall see. These observations are significant in the following sense:

- Since we do not allow name escape, we must keep the α-conversion.
- Since we allow the α-conversion, we must *not* have dynamic fixpoint operation. For the same reason constant definition should be prohibited.

We conclude that in π-calculus constant definitions are problematic.

The purpose of the above discussion is to explain why in π-calculus the parametric definitions [27] are always preferred. A finite set of parametric definitions take the following shape:

$$\begin{array}{rcl} D_1(x_{11},\ldots,x_{1n_1}) & = & E_1, \\ & \vdots & \\ D_n(x_{n1},\ldots,x_{nn_n}) & = & E_n. \end{array}$$

For each $i \in \{1,\ldots,n\}$, the name variables $x_{i1},\ldots,x_{in_i}$ are *parameters* that satisfy the condition $fv(E_i) \subseteq \{x_{i1},\ldots,x_{in_i}\}$. When using the definiendum $D_i(x_{i1},\ldots,x_{in_i})$, the parametric names $x_{i1},\ldots,x_{in_i}$ must be instantiated. For instance the expression $D_i(n_1,\ldots,n_{n_i})$ stands for $E_i\{n_1/x_{i1},\ldots,n_{n_i}/x_{in_i}\}$. The power of parametric definition is due to the fact that, for each $i \in \{1,\ldots,n\}$, the definiendum $D_i(x_{i1},\ldots,x_{in_i})$ may appear in any of $E_1,\ldots,E_n$ in an instantiated form. The following is a concrete example of parametric definition.

$$\begin{array}{rcl} D_1(x,y,z) & = & \overline{x}x \,|\, (a)(\overline{z}b \,|\, D_2(a,b,z)), \\ D_2(x,y,z) & = & \overline{y}y \,|\, (b)(\overline{z}a \,|\, D_1(a,b,z)). \end{array}$$

Notice that the instantiations $D_1(a,b,z)$ and $D_2(a,b,z)$ are respectively the expressions $\overline{a}a \,|\, (a)(\overline{z}b \,|\, D_2(a,b,z))$ and $\overline{b}b \,|\, (b)(\overline{z}a \,|\, D_1(a,b,z))$. Another example of parametric definition is

$$L(x) \quad = \quad (c)(\overline{x}c \,|\, L(c)).$$

Clearly $L(c_0)$ admits the action sequence of (24).

Parametric definitions admit α-conversion. In [27] it is shown that parametric definitions and constant definitions are equivalent in CCS. This result supports the view that constant definitions are abbreviations of parametric definitions.

$$\begin{aligned}
[\![\mathbf{0}]\!] &\overset{\text{def}}{=} \mathbf{0},\\
[\![\sum_{i\in I}\lambda_i.E_i]\!] &\overset{\text{def}}{=} \sum_{i\in I}\lambda_i.[\![E_i]\!],\\
[\![(a)E]\!] &\overset{\text{def}}{=} (a)[\![E]\!],\\
[\![E_1\,|\,E_2]\!] &\overset{\text{def}}{=} [\![E_1]\!]\,|\,[\![E_2]\!],\\
[\![D(n_1,\ldots,n_i)]\!] &\overset{\text{def}}{=} (c_D)(\overline{c_D}(c).\bar{c}n_1.\ldots.\bar{c}n_i\,|\,\mu X.c_D(z).z(x_1).\ldots.z(x_i).[\![E]\!]\varsigma).
\end{aligned}$$

In the fifth line of the definition, we assume that $D(x_1,\ldots,x_i)=E$ is a parametric definition, that $D(z_{11},\ldots,z_{1i})$, …, $D(z_{k1},\ldots,z_{ki})$ are all distinct instantiations of $D(x_1,\ldots,x_i)$ in E, and that the substitution ς is

$$[(\overline{c_D}(c).\bar{c}z_{11}.\ldots.\bar{c}z_{1i}\,|\,X)/[\![D(z_{11},\ldots,z_{1i})]\!],\ldots,(\overline{c_D}(c).\bar{c}z_{k1}.\ldots.\bar{c}z_{ki}\,|\,X)/[\![D(z_{k1},\ldots,z_{ki})]\!]],$$

and that c_D, c are fresh names.

Figure 6: Translation from $\pi^{m\text{def}}$ to π^m

Let π^{def} be π-calculus with parametric definition but without fixpoint operator. Let $\pi^{m\text{def}}$, respectively $\pi^{s\text{def}}$, be obtained from π^{def} by replacing general choice by mixed choice, respectively separated choice. It should not come as a surprise that parametric definition is strong enough to code up fixpoint.

Proposition 16. *$\pi \sqsubseteq^{\pi}_{\downarrow} \pi^{\text{def}}$, $\pi^m \sqsubseteq^{\pi}_{\downarrow} \pi^{m\text{def}}$, $\pi^s \sqsubseteq^{\pi}_{\downarrow} \pi^{s\text{def}}$.*

Proof. Static fixpoints can be easily defined in terms of parametric definitions. The expression $\mu X.E$, in which E does not contain any μ-operator, is translated to the parametric definition

$$X(x_1,\ldots,x_n)=E\{X(x_1,\ldots,x_n)/X\}.$$

The general fixpoint expressions can be translated in an inside-out fashion. □

In the other direction, parametric definition can be simulated by fixpoint operator, which draws a sharp contrast to the situation in CCS. The instantiation of a parametric definition can be achieved by local communication as long as choices are guarded. Milner has used this idea in [44] to code up parametric definitions in $\pi^{m!}$-calculus.

Proposition 17. *$\pi^{m\text{def}} \sqsubseteq^{\pi}_{\downarrow} \pi^m$ and $\pi^{s\text{def}} \sqsubseteq^{\pi}_{\downarrow} \pi^s$.*

Proof. The codivergent subbisimilarity from $\pi^{m\text{def}}$ to π^m is generated by an encoding from the former to the latter, which restricts to an encoding from $\pi^{s\text{def}}$ to π^s. Formally the structural encoding from $\pi^{m\text{def}}$-processes to π^m-processes is defined in Figure 6. Let

$$D(x_1,\ldots,x_i)=E$$

be a parametric definition. For simplicity we assume for the moment that E does not contain any instantiation of any definiendum other than D. In E there might be several instantiations of the definiendum D. To understand the encoding it is useful to think of D as being interpreted as

$$\mu X.c_D(z).z(x_1,\ldots,x_i).[\![E]\!]\varsigma \tag{34}$$

where c_D is a name that can be used to access the definiendum and $[\![E]\!]\varsigma$ is obtained from $[\![E]\!]$ by replacing every instantiation of D by its encoding. The encoding of the instantiation $D(n_1,\ldots,n_i)$ is given by

$$\overline{c_D}(c).\bar{c}n_1.\ldots.\bar{c}n_i \mid \mu X.c_D(z).z(x_1,\ldots,x_i).[\![E]\!]\varsigma. \tag{35}$$

Obviously in (34) the expression (35) is equivalent to $\overline{c_D}(c).\bar{c}n_1.\ldots.\bar{c}n_i \mid X$. Hence the definition in Figure 6.

The intuition about the translation is that every time the fixpoint in (34) is unfolded, the name variables in E are updated via internal communications.

Let's now take a look at the operational aspect of the encoding. First of all notice that in π^m-calculus one has that

$$[\![D(n_1,\ldots,n_i)]\!] \Longrightarrow (c_D)[\![E]\!]\varsigma\{\mu_X/X\}\sigma$$

and

$$[\![D(n_1,\ldots,n_i)]\!] \approx^{\pi} (c_D)[\![E]\!]\varsigma\{\mu_X/X\}\sigma$$

where μ_X stands for $\mu X.c_D(z).z(x_1).\ldots.z(x_i).[\![E]\!]\varsigma$ and σ is $\{n_1/x_1,\ldots,n_i/x_i\}$. It might occur to the reader whether something would go wrong if some occurrence $\overline{c_D}(c).\bar{c}n_1.\ldots.\bar{c}n_i$ communicates with another copy of μ_X rather than the neighboring μ_X. That such a communication would do no harm is guaranteed by the following property:

> In π^m-calculus, $(c_D)[\![E]\!]\varsigma\{\mu_X/X\}\{n_1/x_1,\ldots,n_i/x_i\}$ is structural bisimilar to $[\![E\{n_1/x_1,\ldots,n_i/x_i\}]\!]$.

Now define the relation $\mathcal{R}$ from $\pi^{m\mathrm{def}}$-calculus to π^m-calculus as follows:

$$\mathcal{R} \stackrel{\mathrm{def}}{=} \{(P,[\![P]\!]) \mid P \text{ is a } \pi^{m\mathrm{def}} \text{ process}\}.$$

Let $\approx^{\pi}_{\downarrow}$ be the largest codivergent π-bisimulation. The composition $\mathcal{R};\approx^{\pi}_{\downarrow}$ satisfies the following properties:

- It is codivergent.
- Suppose $P_1\ \mathcal{R}\ [\![P_1]\!] \approx^{\pi}_{\downarrow} Q_1$. If $P_1 \xrightarrow{\lambda} P_2$ then $[\![P_1]\!] \stackrel{\lambda}{\Longrightarrow} P' \approx^{\pi}_{\downarrow} [\![P_2]\!]$. By definition some P'' exists such that $Q_1 \stackrel{\widehat{\lambda}}{\Longrightarrow} Q_1' \approx^{\pi}_{\downarrow} P'$. Therefore $[\![P_2]\!]\ \mathcal{R} \approx^{\pi}_{\downarrow} Q_1'$. In the other direction if $Q_1 \xrightarrow{\lambda} Q_1'$ then P' exists such that $[\![P_1]\!] \stackrel{\widehat{\lambda}}{\Longrightarrow} P' \approx^{\pi}_{\downarrow} Q_1'$. It can be proved that some P_2 exists such that $P_1 \stackrel{\widehat{\lambda}}{\Longrightarrow} P_2$ for some P_2 such that $[\![P_2]\!] \approx^{\pi}_{\downarrow} P'$. Hence $P_1'\ \mathcal{R} \approx^{\pi}_{\downarrow} Q_1'$. So the relation $\mathcal{R};\approx^{\pi}_{\downarrow}$ satisfies the bisimulation property.

$$\begin{array}{rcl}
D_{\mathbf{0}}(x) & = & \overline{x}, \\
D_{\tau.E}(x\widetilde{z}) & = & \overline{x}+\tau.(c)D_E(c\widetilde{z}), \\
D_{\overline{p}q.E}(x\widetilde{z}) & = & \overline{x}+\overline{p}q.(c)D_E(c\widetilde{z}), \\
D_{p(v).E}(x\widetilde{z}) & = & \overline{x}+p(v).(c)D_E(c\widetilde{z}v), \\
D_{(a)E}(x\widetilde{z}) & = & (a)D_E(x\widetilde{z}a), \\
D_{E_1+E_2}(x\widetilde{z}) & = & \overline{x}+\tau.(c)(D_{E_1}(c\widetilde{z}) \mid c.D_{E_1+E_2}(x\widetilde{z}))+\tau.(c)(D_{E_2}(c\widetilde{z}) \mid c.D_{E_1+E_2}(x\widetilde{z})), \\
D_{E_1 \mid E_2}(x\widetilde{z}) & = & \overline{x}+\tau.(c)(V_{E_1|E_2}(xc\widetilde{z}) \mid W_{E_1|E_2}(xc\widetilde{z})), \\
V_{E_1|E_2}(xu\widetilde{z}) & = & \tau.(c)(D_{E_1}(c\widetilde{z}) \mid c.V_{E_1|E_2}(xu\widetilde{z}))+u.D_{E_1 \mid E_2}(x\widetilde{z}), \\
W_{E_1|E_2}(xu\widetilde{z}) & = & \tau.(c)(D_{E_2}(c\widetilde{z}) \mid c.W_{E_1|E_2}(xu\widetilde{z}))+\overline{u},
\end{array}$$

where c is a fresh name.

Figure 7: Transformation from π^{def} to $\pi^{m\mathrm{def}}$.

We conclude that $\mathcal{R};\approx^{\pi}_{\downarrow}$ is a subbisimilarity. $\square$

Having seen several encodings, it should now become a routine to give an encoding from π^{def}-calculus to $\pi^{m\mathrm{def}}$-calculus. First of all we need to define a transformation from π^{def}-expressions containing no parametric definitions to $\pi^{m\mathrm{def}}$-expressions. This encoding is defined in Figure 7. It copies the idea of the encoding of CCS into CCS^m. Next we generalize it to the processes containing parametric definitions. Suppose the following definitions are in π^{def}-calculus:

$$\begin{array}{rcl}
C_1(\widetilde{x_1}) & = & P_1, \\
 & \vdots & \\
C_i(\widetilde{x_i}) & = & P_i.
\end{array}$$

Their translations are defined as follows:

$$\begin{array}{rcl}
D_{C_1}(\widetilde{x_1}) & = & D_{P_1}, \\
 & \vdots & \\
D_{C_i}(\widetilde{x_i}) & = & D_{P_i},
\end{array}$$

where an instantiation $C_j(\widetilde{n})$ on the right hand sides of these definitions is translated to $D_{C_j}(\widetilde{n})$. The verification that such an encoding is correct follows the same argument given in the proof of Proposition 10.

Proposition 18. $\pi^{\mathrm{def}} \sqsubseteq^{\pi} \pi^{m\mathrm{def}}$.

3.5 Interaction Hierarchy of Pi

We have now obtained all the subbisimilarity relationships demonstrated in the following reasoning:

$$\begin{array}{rll}
\pi^{!} & \sqsubseteq^{\pi}_{\downarrow} & \pi \\
 & \sqsubseteq^{\pi}_{\downarrow} & \pi^{\mathrm{def}} \\
 & \sqsubseteq^{\pi} & \pi^{m\mathrm{def}} \\
 & \sqsubseteq^{\pi} & \pi^{s\mathrm{def}} \\
 & \sqsubseteq^{\pi}_{\downarrow} & \pi^{s} \\
 & \sqsubseteq^{\pi}_{\downarrow} & \pi^{m} \\
 & \sqsubseteq^{\pi}_{\downarrow} & \pi^{m!} \\
 & \sqsubseteq^{\pi}_{\downarrow} & \pi^{s!} \\
 & \sqsubseteq^{\pi}_{\downarrow} & \pi^{!}.
\end{array}$$

The propositions involved in the above inference are Proposition 18, Proposition 17, Proposition 16 and Proposition 14. These subbisimilarity relationships lead to the first of the two main results of this section.

Theorem 3. *All the nine π-variants are interactively equivalent.*

Theorem 3 reinforces our belief that communications in π-calculus is far more robust than the interactions of CCS. The name passing mechanism is so strong that a restriction of it often would not reduce its interactive power and that it often uplifts a weak form of an operator to be just as expressive interactively as a strong form of the operator.

Having said that, different π-variants do differ if Computational Criterion is taken into consideration. The following theorem states the second main results of the section.

Theorem 4. *The codivergent subbisimilarity relationships between the nine π-variants are depicted in Figure 8.*

Proof. The negative result $\pi^{!} \not\sqsubseteq^{\pi}_{\downarrow} \pi^{m!}$ can be promoted to $\pi \not\sqsubseteq^{\pi}_{\downarrow} \pi^{m}$ and $\pi^{\mathrm{def}} \not\sqsubseteq^{\pi}_{\downarrow} \pi^{m\mathrm{def}}$. □

The nine π-variants can be classified to at most five equivalence classes. Mixed choice, and separated choice as well, forces the three forms of recursion to be equivalent.

The only open problem left for further study is stated next.

Problem 1. *$\pi^{\mathrm{def}} \sqsubseteq^{\pi}_{\downarrow} \pi$?*

Consider *Local Random Timer* defined as follows:

$$LTimer(x) \overset{\mathrm{def}}{=} \overline{a}a + (c)(\overline{x}c \,|\, LTimer(c)).$$

$$\begin{tikzcd}
\pi^{\mathrm{def}} \arrow[r, tail, dashed, shift left, "?"] \arrow[d, tail, shift right, "/" marking] & \pi \arrow[l, tail, shift left] \arrow[r, tail, shift left, "\backslash" marking] \arrow[d, tail, shift right, "/" marking] & \pi^{!} \arrow[l, tail, shift left] \arrow[d, tail, shift right, "/" marking] \\
\pi^{m\mathrm{def}} \arrow[u, tail, shift right] \arrow[r, tail, shift left] & \pi^{m} \arrow[u, tail, shift right] \arrow[l, tail, shift left] \arrow[r, tail, shift left] \arrow[d, tail, shift right, "/" marking] & \pi^{m!} \arrow[u, tail, shift right] \arrow[l, tail, shift left] \\
\pi^{s\mathrm{def}} \arrow[r, tail, shift left] & \pi^{s} \arrow[u, tail, shift right] \arrow[l, tail, shift left] \arrow[r, tail, shift left] & \pi^{s!} \arrow[l, tail, shift left]
\end{tikzcd}$$

Figure 8: Codivergent Subbisimilarities of Pi

This process is in π^{def}-calculus. It has the following operational behavior:

$$LTimer(c_0) \xrightarrow{\overline{c_0}(c_1)} \ldots \xrightarrow{\overline{c_i}(c_{i+1})} \xrightarrow{\overline{a}a} \xrightarrow{\overline{c_{i+1}}(c_{i+2})} \ldots \xrightarrow{\overline{c_j}(c_{j+1})} \mathbf{0}$$

for any chosen $j \geq i \geq 0$. An interesting operational property of Local Random Timer is demonstrated in the following transition:

$$LTimer(c_0) \xrightarrow{c_0(c_1)} LTimer(c_1) \sim LTimer(c_0)\{c_1/c_0\}.$$

We conjecture that there is no π-process that can bisimulate $LTimer(c_0)$ in a non-divergent way. Although the conjecture has not been proved, the following remarks might be useful for further investigation. Assume that there were a π-process P that bisimulated $LTimer(c_0)$ and that P cannot do any τ-action. Here are some observations about P:

1. Since P can perform both the $\overline{a}a$-action and the $\overline{c_0}(c_1)$-action, it cannot be in prefix form.
2. Suppose that P is a composition $P_0 \mid P_1$. Then either $P_0 \sim \mathbf{0}$ or $P_1 \sim \mathbf{0}$. Otherwise the operational behavior is different from that of $LTimer(c_0)$. So we may assume that P is not a composition.
3. Suppose that P is a summation P_0+P_1. Without loss of generality, we may assume that $P_0 \xrightarrow{c_0(c_1)}\Longrightarrow P_0' \approx LTimer(c_1)$. Hence $P_0'\{c_0/c_1\} \approx LTimer(c_0) \approx P$.
4. There must be a subexpression $\overline{a}a.A$ of P underneath a μ-operator. Otherwise there would be only a finite number of $\overline{a}a$-derivatives.
5. There must be a subexpression of the form $\mu X.E$ such that there is an unguarded occurrence of X in E. Otherwise there would be only a finite number of $\overline{a}a$-derivatives.

6. There must be a subexpression of the form $\mu X.E$ such that there is a subexpression $\overline{a}a.A$ of E and that there is an unguarded occurrence of X in E. Moreover $\overline{a}a.A$ and the unguarded occurrence of X must not be in a concurrent position.

7. No reachable occurrence of X in $\mu X.E$ can be in concurrent position with the unguarded occurrence of X. Otherwise P would be able to do two $\overline{a}a$ actions sequentially. Such unreachable occurrences of X can be removed.

The intuition is that after action $\overline{c_i}(c_{i+1})$, a component of the residual of the action must pass c_{i+1} to another concurrent component via a local name. The above observations suggest that action $\overline{a}a$ cannot maintain this capability while setting the timer randomly.

4 Application of Subbisimilarity

In this section we put into use the results obtained in Section 2 and Section 3. We shall be discussing two issues. One is about the independence of the operators of π-calculus. This problem cannot be tackled without any results on expressiveness. The other is about choosing a representative among the variants of a process calculus. This is made easier by the results on expressiveness.

4.1 Operator Independence

An important application of a theory of expressiveness is to formally establish the independence of process operators. In this section we take π-calculus as an example to explain the general approach.

Let π_0 be a π-variant and π_0^{-op} be obtained from π_0 by removing the operator *op* and the associated operational rules from π_0. We say that the operator *op* is *independent* to the other operators of π_0 if $\pi_0 \not\sqsubseteq^{\pi} \pi_0^{-op}$.

The name passing communication mechanism of π-calculus makes possible the internalization of the conditionals useful in programming. It is interesting to see how the conditionals enhance the interactability of π-calculus. The syntaxes of the match and the mismatch operators are $[p{=}q]E$ and $[p{\neq}q]E$ respectively. The semantics is defined by the following rules.

$$\frac{E \xrightarrow{\lambda} E'}{[a{=}a]E \xrightarrow{\lambda} E'} \qquad \frac{E \xrightarrow{\lambda} E'}{[a{\neq}b]E \xrightarrow{\lambda} E'}$$

We use the notations $\pi^{=}$, $\pi^{\neq}$ and π^{c} to stand for the π-variant with respectively the match operator $=$, the mismatch operator $\neq$ and both the match and mismatch operators. In view of Theorem 3, we assume that the recursion capacity of π^c ($\pi^{=}$, $\pi^{\neq}$) is provided by replication. It is clear that $\pi \sqsubseteq^{\pi} \pi^{=}$ ($\pi^{\neq}$) $\sqsubseteq^{\pi} \pi^{c}$. More about the interactability of the conditionals are given next.

Proposition 19. *The following statements are valid.*
(i) $\pi^{=} \not\sqsubseteq^{\pi} \pi$ *and* $\pi^{=} \not\sqsubseteq^{\pi} \pi^{\neq}$.

(ii) $\pi^{\neq} \not\sqsubseteq^{\pi} \pi$ *and* $\pi^{\neq} \not\sqsubseteq^{\pi} \pi^{=}$.
(iii) $\pi^{c} \not\sqsubseteq^{\pi} \pi^{=}$ *and* $\pi^{c} \not\sqsubseteq^{\pi} \pi^{\neq}$.

Proof. (i) Consider $a(x).[x{=}c]\overline{b}b$. Suppose $a(x).[x{=}c]\overline{b}b\ \mathcal{R}\ P$ for a π-process P and a subbisimilarity $\mathcal{R}$ from $\pi^{=}$ to π. The two consecutive observable actions $a(x).[x{=}c]\overline{b}b \xrightarrow{ac} [c{=}c]\overline{b}b \xrightarrow{\overline{b}b} \mathbf{0}$ must be simulated by P in the manner $P \Longrightarrow P_1 \xrightarrow{ac} P_2 \Longrightarrow P_3 \xrightarrow{\overline{b}b} P_4 \Longrightarrow\approx \mathbf{0}$. We claim that for a fresh d some P_2', P_3', P_4' exist such that

$$P_1 \xrightarrow{ad} P_2' \Longrightarrow P_3' \xrightarrow{\overline{b}b} P_4'. \tag{36}$$

This fact can be proved by induction. The tricky part is to do with prefix operator. Suppose without loss of generality that $P_1 \equiv a(x).P_1'$. Then clearly $P_1'\{c/x\} \equiv P_2$ and $P_1'\{d/x\} \equiv P_2'$. It should be clear that $P_2 \approx \overline{b}b$. This equivalence implies that the internal actions in $P_2 \Longrightarrow P_3$ must be via local names. Since P_2 does not contain any match operations, the substitution $\{c/x\}$ does not enable any internal actions in P_1'. Therefore $P_1' \Longrightarrow\xrightarrow{\overline{b}b}$, from which (36) follows. But (36) contradicts the fact that $a(x).[x{=}c]\overline{b}b \xrightarrow{ad} [d{=}c]\overline{b}b$. Hence $\pi^{=} \not\sqsubseteq^{\pi} \pi$. The argument for $\pi^{=} \not\sqsubseteq^{\pi} \pi^{\neq}$ is similar.

(ii) Suppose $a(x).[x{\neq}c]\overline{b}b$ were equivalent to a process P of π ($\pi^{=}$). For a fresh e the two consecutive actions $a(x).[x{\neq}c]\overline{b}b \xrightarrow{ae} [e{\neq}c]\overline{b}b \xrightarrow{\overline{b}b} \mathbf{0}$ must be simulated by $P \overset{ae}{\Longrightarrow}\overset{\overline{b}b}{\Longrightarrow}$. But then $P \overset{ac}{\Longrightarrow}\overset{\overline{b}b}{\Longrightarrow}$, which cannot be simulated by $a(x).[x{\neq}c]\overline{b}b$ of course.

(iii) The proofs are similar. □

Theorem 5. *All the operators of* π^{c}*-calculus are independent.*

Proof. Prefix operator and fixpoint operator are easily seen to be independent from the other operators. The proof of Proposition 11 shows that choice operator is independent. Proposition 19 says that match, and mismatch as well, is independent from the rest of the operators of π^{c}-calculus.

Localization operator is independent because restricted output actions are interactively different from input actions and free output actions. We will go for a formal proof of this fact. Consider $Counter^{\pi}$. It is easy to check that only after performing equal numbers of *inc*'s and *dec*'s can $Counter^{\pi}$ do a *zero* action. For instance the following is an admissible sequence of actions:

$$Counter^{\pi} \underbrace{\overset{inc}{\Longrightarrow} \ldots \overset{inc}{\Longrightarrow}}_{i \text{ times}}\underbrace{\overset{dec}{\Longrightarrow} \ldots \overset{dec}{\Longrightarrow}}_{i \text{ times}} \xrightarrow{zero} .$$

Let π^{-L} be obtained from π-calculus by removing the localization operator. Now π^{-L} does not have any bound output actions. A translation from π to π^{-L} must explain how the observable actions of the former are translated to the latter. In other words we need to specify a map of the following type.

$$\iota : \{ac, \overline{a}c, \overline{a}(c) \mid a, c \in \mathcal{N}\} \to \{ac, \overline{a}c \mid a, c \in \mathcal{N}\}.$$

In the present proof we do not have to assume anything for ι, which makes our negative result really strong. We prove that π-calculus cannot be encoded in π^{-L}-calculus by showing that there is no π^{-L}-process that can demonstrate the operational behavior of $Counter^{\pi}$ under any map ι whatsoever. Let us say that a π^{-L}-process P_0 has *Counting Property* if there exist pairwise distinct $\mathsf{a}, \mathsf{b}, \mathsf{c} \in \mathcal{N} \cup \overline{\mathcal{N}}$ such that following four conditions hold.

- The following infinite action sequence is admissible

$$P_0 \overset{\mathsf{a}d_0}{\Longrightarrow} P_1 \overset{\mathsf{a}d_1}{\Longrightarrow} \ldots P_i \overset{\mathsf{a}d_i}{\Longrightarrow} \ldots . \tag{37}$$

- If $P_0 \overset{\mathsf{a}d_0}{\Longrightarrow} \ldots \overset{\mathsf{a}d_{i-1}}{\Longrightarrow} P_i$ then $e_0, \ldots, e_{i-1}, f_0$ exist such that $P_i \overset{\mathsf{b}e_0}{\Longrightarrow} \ldots \overset{\mathsf{b}e_{j-1}}{\Longrightarrow} P_{i+j} \xrightarrow{\mathsf{c}f_0}$ for some P_{i+j}.

- The action sequence (38) is admissible only if $i = j$.

$$P_0 \overset{\mathsf{a}d_0}{\Longrightarrow} \ldots \overset{\mathsf{a}d_{i-1}}{\Longrightarrow} \overset{\mathsf{b}e_0}{\Longrightarrow} \ldots \overset{\mathsf{b}e_{j-1}}{\Longrightarrow} P_{i+j} \xrightarrow{\mathsf{c}f_0} . \tag{38}$$

- P_0 can never ever perform an action $\mathsf{d}d$ such that $\mathsf{d} \notin \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$.

For such a P_0, notice that in (37) the set of the object names $\{d_0, d_1, \ldots, d_i, \ldots\}$ could be infinite. But (37) corresponds to an infinite sequence

$$P_0 \equiv P'_0 \overset{\mathsf{a}d'_0}{\Longrightarrow} P'_1 \overset{\mathsf{a}d'_1}{\Longrightarrow} \ldots P'_i \overset{\mathsf{a}d'_i}{\Longrightarrow} \ldots \tag{39}$$

where $\{d'_0, d'_1, \ldots, d'_i, \ldots\}$ is finite. The claim is based on two observations:

- A subexpression $a(z).A$ of P_i can be activated only if $a \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. A subexpression $\overline{a}z.A$ of P_i can be activated only if $\overline{a} \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. Consequently if $a(z).A$ is a subexpression of P_i and $z(x).B$ (or $\overline{z}y.B$) is a subexpression of A, then $z(x).B$ (or $\overline{z}y.B$) is not fireable unless z is instantiated by a name that appears in $\mathsf{a}, \mathsf{b}, \mathsf{c}$.

- Suppose there are k distinct match or mismatch operators in P_0. In addition to $gn(P_0)$, we only need a set $\mathcal{B}$ of $2k+1$ fresh names. Without loss of generality, we may always assume that

$$\mathcal{B} \cap \{d_0, d_1, \ldots, d_i, \ldots\} = \emptyset.$$

  Now we replace the object names in (37) inductively. If $\mathsf{a}d_0$ is an output action or $d_0 \in gn(P_0) \cup \mathcal{B}$ then do nothing, otherwise choose a name d'_0 in $\mathcal{B}$ that does not change the validity of the conditions in the immediate descendent of the action $\mathsf{a}d_0$. This is possible since there are only k distinct matches and/or mismatches. Let P'_1 be $P_1\{d'_0/d_0\}$. We then continue with

$$P'_1 \overset{\mathsf{a}d_1\{d'_0/d_0\}}{\Longrightarrow} P_2\{d'_0/d_0\} \overset{\mathsf{a}d_2\{d'_0/d_0\}}{\Longrightarrow} \ldots P_i\{d'_0/d_0\} \overset{\mathsf{a}d_i\{d'_0/d_0\}}{\Longrightarrow} \ldots .$$

  In this way we obtain an action sequence like (39).

By extending the definition of the concurrent subexpression to take into account of the conditionals, we can modify Lemma 44 to show that (39) is well quasi-ordered with Compatibility Property. Consequently $P'_i \preceq P'_j$ for some $i < j$. Therefore the following action sequence

$$P'_0 \overset{\mathsf{a}d'_0}{\Longrightarrow} \ldots \overset{\mathsf{a}d'_{j-1}}{\Longrightarrow} P'_j \overset{\mathsf{a}d''_0}{\Longrightarrow} \ldots \overset{\mathsf{a}d''_{i-1}}{\Longrightarrow} \overset{\mathsf{b}e''_0}{\Longrightarrow} \ldots \overset{\mathsf{b}e''_{i-1}}{\Longrightarrow} \xrightarrow{\mathsf{c}f''_0}$$

is admissible, which contradicts (38).

Without localization operator, we cannot even define the counter in π-calculus. We conclude that there is no encoding from π-calculus to π^{-L}-calculus that preserves interactability. □

The crux of the matter is that the standard localization operator provides a mechanism to generate new names. This is the essential difference between π-calculus and the value-passing calculus based on CCS. In the framework of π-calculus, it is the localization operator that gives rise to Turing computability.

Proposition 20. *π^{-L}-calculus is not Turing complete.*

Proof. The termination property in π^{-L}-calculus is decidable. An algorithm that checks this property works as follows: Given a π^{-L}-process P, the number of the immediate τ-actions P may perform is finite. Using this finite branching property we can generate the τ-action tree of P in a breadth first fashion. The nodes of the tree are processes and the edges are labeled by τ. Upon the generation of a node B, it is checked if there is an ancestor A of B such that $A \preceq B$. It is easy to see that there is a recursive algorithm to decide $\preceq$. The generation of the tree terminates with two outcomes. If the tree is finite, the algorithm terminates with answer "yes". Otherwise a node B must be found such that $A \preceq B$ for some ancestor A of B. In this case the algorithm terminates with answer "no". The correctness of the algorithm is guaranteed by the fact that an infinite τ-action sequence in π^{-L}-calculus is well quasi-ordered by Lemma 44.

If π^{-L}-calculus were Turing complete, then according to the condition (5) given after Definition 6 there would have been an algorithm that translates every Deterministic Turing Machine to a π^{-L}-process. By the conditions (3,4) after Definition 6, the translation satisfies the property that a Deterministic Turing Machine is terminating if and only if its interpretation in π^{-L}-calculus is terminating, the latter being decidable. It follows that one could design an algorithm for the halting problem. □

4.2 Model Assessment

A theory of expressiveness should help us in finding an appropriate form of a process calculus among all of its variants. Since we have investigated the expressiveness of CCS-variants and π-variants, we can say something about their candidacies.

4.2.1 CCS with Sufficient Computing Power

Lemma 36 implies some decidability results for CCS. The decidability of the termination property of CCS$^!$ is established in [7]. That result can be strengthened significantly.

Proposition 21. *In* CCS *it is decidable to check if a process admits an infinite action sequence.*

Proof. The expansion order for CCS is a well quasi-order that satisfies the compatibility property. So a proof of the proposition is essentially given in the proof of Proposition 20. Given a CCS process P, the algorithm traverses the action tree of $\mathcal{I}^!(\mathcal{I}_a(P))$, where a is fresh. Upon generating a node P' the algorithm checks if there is an ancestor P'' of P' such that the path from P'' to P' consists of only a-actions and the number of the consecutive a's is $\mathbb{N}_a(P'')$. If the answer is yes, then P' is marked and the algorithm will overlook all the derivatives of P'. □

A practical implication of Proposition 21 is that none of CCS^m, CCS$^!$ and CCS can code up all Turing machines. The proof of this fact is the same as that of Proposition 20.

Theorem 6. *None of* CCS$^!$*,* CCS^m *and CCS is Turing complete.*

For both theoretical and practical reasons, a process calculus that is not Turing complete is undesirable. It follows that CCS$^{\mathrm{def}}$ and CCS$^{m\mathrm{def}}$ are preferable to any of CCS$^!$, CCS^m and CCS. Interactively CCS$^{\mathrm{def}}$ and CCS$^{m\mathrm{def}}$ are equivalent. The latter has an edge over the former since the bisimilarity of CCS$^{m\mathrm{def}}$ is a congruence. We conclude that CCS with constant definitions and guarded choices should be chosen in practice.

4.2.2 Pi with Simple Semantics

Now what do our results on the expressiveness of π-variants tell us about candidacy? Theorem 3 assures us that each of the nine variants can be chosen without losing expressive power. But the diagram in Figure 8 says something more.

- It is more convenient to program in the variants with mixed choice than in the ones with separated choice. A simple program like counter is naturally designed using mixed choice.

- The convenience offered by the variants with unrestricted choice is mainly to do with infinite branching. Theoretically finite branching is preferred. Practically infinite choice can be simulated by mixed choice anyway. Moreover the unrestricted choice operator is not always a congruence operator. Mixed choice does not have that problem.

It seems that having mixed choice in π is a good choice. Which of the three variants with mixed choice do we prefer? Well if we do not go into the higher order scenario, we should prefer $\pi^{m!}$ since it is a purely first order calculus.

In both theory and practice, match operator is frequently necessary. It allows one to write **if_then_** commands in π-calculus and makes it possible to state an expansion law for π-calculus. If we put match operator in the repertoire of π-calculus, we get nine variants with match combinator. In view of Theorem 5, the operator strictly increases the expressive power. An interesting question is if Theorem 3 and Theorem 4 hold for the nine new variants. Our preliminary investigation seems affirmative. But details remain to be checked. Let $\pi^{!=}$ be $\pi^{!}$ extended with match operator. In $\pi^{!=}$ the following structural equalities hold:

$$\begin{aligned}
[p{=}q]\sum_{i\in I}\lambda_i.P_i &\sim \sum_{i\in I}[p{=}q]\lambda_i.P_i,\\
[p{=}q](P\,|\,P') &\sim [p{=}q]P\,|\,[p{=}q]P',\\
[p{=}q](a)P &\sim (a)[p{=}q]P,\\
[p{=}q]!P &\sim ![p{=}q]P.
\end{aligned}$$

These equalities strongly suggest the following alternative syntax of $\pi^{!=}$-calculus:

$$P := \sum_{i\in I}\varphi_i\lambda_i.P_i \mid P\,|\,P' \mid (a)P \mid !P, \tag{40}$$

where φ_i is a finite conjunction of matches. Now actions are conditioned; but programs are not. Let π^{if} denote this variant. The following fact is obvious.

Proposition 22. $\pi^{if} \sqsubseteq^{\pi}_{\downarrow} \pi^{!=} \sqsubseteq^{\pi}_{\downarrow} \pi^{if}$.

Proof. $\pi^{if} \sqsubseteq^{\pi}_{\downarrow} \pi^{!=}$ is obvious. The proof of Lemma 37 can be carried out in the present setting. A $\pi^{!=}$-process is structurally bisimilar to a $\pi^{m=}$-process of the form $\sum_{i\in I}\varphi_i\lambda_i.P_i$, which can be bisimulated by a π^{if}-process. So we also have $\pi^{!=} \sqsubseteq^{\pi}_{\downarrow} \pi^{if}$. □

If π^{if}-calculus is further promoted with **if_then_else** capacity, it need to incorporate mismatch operator. But how should the combinator be incorporated? The answer is to stick to the syntax of (40). Let π^{case} be this calculus. Let $\pi^{!c}$ be $\pi^{!}$-calculus extended with the conditionals. The following fact is assuring.

Proposition 23. $\pi^{case} \sqsubseteq^{\pi}_{\downarrow} \pi^{!c} \sqsubseteq^{\pi}_{\downarrow} \pi^{case}$.

Proof. The proof is the same as that of Proposition 22. □

The above proposition suggests that we may as well choose π^{case} as *the* π-calculus.

The exercises carried out in this and previous subsections intend to convey the following picture: When designing a process calculus, we are often given a lot of candidates. We should evaluate the candidates according to some criteria. The criteria may concern pragmatics, algebraic beauty, programming style etc.. Formal expressiveness results should play a key role in some design decisions. The final design should also be evaluated in terms of expressiveness.

5 More Subbisimilarities

In this section we apply the technique of subbisimilarity to study expressiveness for some π-related calculi. The π-calculus with dynamic binding has a localization operator that behaves like a physical ambient. The value-passing CCS is closer to CCS than to π-calculus. The asynchronous π-calculus has a quite different observational theory from π-calculus. The internal π-calculus stays right in between CCS and π-calculus.

5.1 Name Escape Communication

An alternative to the *name extrusion* rule

$$\frac{E \xrightarrow{\overline{a}c} E'}{(c)E \xrightarrow{\overline{a}(c)} E'}$$

is the following *name escape* rule.

$$\frac{E \xrightarrow{\overline{a}c} E'}{(c)E \xrightarrow{\overline{a}c} (c)E'}$$

Let π^d be π-calculus with the name escape. In π^d-calculus one has typically following communication

$$a(z).P \,|\, (c)(R \,|\, \overline{a}c.Q) \xrightarrow{\tau} P\{c/z\} \,|\, (c)(R \,|\, Q).$$

It is clear from the above communication that α-conversion must be banned in π^d. One of its implications is that in π^d-calculus name capture must be admissible. So we should have

$$(c)(R \,|\, a(z).P) \,|\, \overline{a}c.Q \xrightarrow{\tau} (c)(R \,|\, P\{c/z\}) \,|\, Q.$$

A further consequence is that in π^d-calculus fixpoint operator is dynamic. A minute's thought would lead the reader to conclude that π^d-calculus is strictly more powerful than $\pi^{d!}$-calculus, where $\pi^{d!}$ stands for the π^d-calculus with replication instead of dynamic fixpoint operator.

In π^d-calculus as well as $\pi^{d!}$-calculus there is no bound output. We leave the definition of the subbisimilarity $\sqsubseteq^d$ to the reader.

Proposition 24. $\pi^{d!} \sqsubseteq^d_{\downarrow} \pi^d \not\sqsubseteq^d \pi^{d!}$.

Proof. In π-calculus with dynamic binding, localization operator is never relocated. We could define an expansion order $\preceq^{\pi^{d!}}$ in the standard manner. If the infinite action sequence

$$P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} P_2 \xrightarrow{\lambda_2} \ldots \tag{41}$$

in $\pi^{d!}$-calculus satisfies the condition that the set

$$\{c \mid \exists i{\geq}0.\exists a{\in}\mathcal{N}.\lambda_i = ac\} \tag{42}$$

is finite, then the conditions of Lemma 44 are met. In this case (41) is well quasi-ordered. If the condition (42) does not hold, we can obtain an infinite action sequence from (41), as we have done in the proof of Theorem 5, such that (42) is satisfied by the new action sequence. In either case we can show that no $\pi^{d!}$-process satisfies Counting Property. On the other hand CCS$^{\mathrm{def}}$ can be embedded in π^d-calculus. So the counter can be defined in π^d-calculus. □

The following corollary is implied by the above proof.

Corollary 8. *$\pi^{d!}$-calculus is not Turing complete.*

It follows that $\pi^{d!}$ -calculus has to be abandoned. However π^d-calculus is a natural extension of CCS$^{\mathrm{def}}$ and is quite interesting.

5.2 Value-Passing CCS

In π-calculus the content of a communication is nothing but names, whereas in value-passing CCS the content of a communication may contain anything but names. The value-passing CCS [39, 33, 36], CCS$_{vp}$ for short, makes use of two oracles, a data algebra $\mathfrak{D}$ and a boolean algebra $\mathfrak{B}$. The algebra $\mathfrak{B}$ is the propositional logic with the equality judgment $=$ on data expressions of $\mathfrak{D}$. The set of the data expressions $\mathfrak{Dexp}$, ranged over by $e, e', \ldots$ is constructed from $\mathfrak{D}$ and the set of the data variables $\mathfrak{Dvar}$. The set $\mathfrak{Bexp}$ of the boolean expressions, ranged over by $\phi, \phi', \ldots$ is constructed from $\mathfrak{B}$ and the set $\mathfrak{Bvar}$ of the boolean variables.

The following is the BNF grammar of CCS$_{vp}$ expressions

$$E \quad := \quad \mathbf{0} \mid X \mid \lambda.E \mid E \,|\, E' \mid (a)E \mid \mathbf{if}\ \phi\ \mathbf{then}\ E \mid E{+}E' \mid \mu X.E,$$

where $\lambda \in \{a(x), \overline{a}(e) \mid x \in \mathfrak{Dvar}, e \in \mathfrak{Dexp}\} \cup \{\tau\}$ and $\phi \in \mathfrak{Bexp}$. Depending on how we deal with data and boolean expressions, the operational semantics of CCS$_{vp}$ can be classified into two types. The *symbolic semantics* [33, 36] is based on a proof theory of data and boolean domains. The *concrete semantics* [39] is built on a model theory of data and boolean domains. The evaluation function $[\![\_]\!]$ returns the value of an expression e (or ϕ) whenever e (or ϕ) contains neither domain variables nor boolean variables. The concrete semantics of CCS$_{vp}$ is defined by the following rules.

Prefix

$$\frac{}{\tau.E \xrightarrow{\tau} E} \qquad \frac{}{a(x).E \xrightarrow{a(v)} E\{v/x\}} \qquad \frac{[\![e]\!] = v}{\overline{a}(e).E \xrightarrow{\overline{a}(v)} E}$$

Composition

$$\frac{E \xrightarrow{\lambda} E'}{E \,|\, F \xrightarrow{\lambda} E' \,|\, F} \qquad \frac{E \xrightarrow{a(v)} E' \qquad F \xrightarrow{\overline{a}(v)} F'}{E \,|\, F \xrightarrow{\tau} E' \,|\, F'}$$

Restriction

$$\frac{E \xrightarrow{\lambda} E' \qquad a \text{ does not appear in } \lambda}{(a)E \xrightarrow{\lambda} (a)E'}$$

Condition

$$\frac{E \xrightarrow{\lambda} E' \quad [\![\phi]\!] = \top}{\textbf{if } \phi \textbf{ then } E \xrightarrow{\lambda} E'}$$

Choice

$$\frac{E \xrightarrow{\lambda} E'}{E{+}F \xrightarrow{\lambda} E'}$$

Recursion

$$\frac{E\{\mu X.E/X\} \xrightarrow{\lambda} E'}{\mu X.E \xrightarrow{\lambda} E'}$$

Without assuming too much on the domain and boolean algebras, one may define in CCS$_{vp}$ the counter as follows:

$$\begin{aligned}
C_{vp} &\stackrel{\text{def}}{=} (c)(\overline{c}0 \mid \mu X.c(x).(Z_{vp} + O_{vp})),\\
Z_{vp} &\stackrel{\text{def}}{=} \textbf{if } x{>}0 \textbf{ then } (I_{vp} + D_{vp}),\\
O_{vp} &\stackrel{\text{def}}{=} \textbf{if } x{=}0 \textbf{ then } (I_{vp} + \overline{zero}(0).(X \mid \overline{c}0)),\\
I_{vp} &\stackrel{\text{def}}{=} \overline{inc}(1).(X \mid \overline{c}(x{+}1)),\\
D_{vp} &\stackrel{\text{def}}{=} \overline{dec}(1).(X \mid \overline{c}(x{-}1)).
\end{aligned}$$

The counter can increment by performing $\overline{inc}(1)$, decrement by performing $\overline{dec}(1)$, and indicate that the current value is zero by performing $\overline{zero}(0)$. Notice that this example can be defined using replication.

We shall define subbisimilarities between CCS$_{vp}$ variants in the style of concrete semantics.

Definition 17. *A subbisimilarity $\mathcal{R}$ from* CCS$^1_{vp}$ *to* CCS$^2_{vp}$ *is a total relation $\mathcal{R}$ from $\mathcal{P}_{\text{CCS}^1_{vp}}$ to $\mathcal{P}_{\text{CCS}^2_{vp}}$ such that the following statements are valid.*

1. *If $Q\mathcal{R}^{-1}P \xrightarrow{\lambda}_1 P'$ then $Q \stackrel{\widehat{\lambda}}{\Longrightarrow}_2 Q'\mathcal{R}^{-1}P'$ for some Q'.*

2. *If $P\mathcal{R}Q \xrightarrow{\lambda}_2 Q'$ then $P \stackrel{\widehat{\lambda}}{\Longrightarrow}_1 P'\mathcal{R}Q'$ for some P'.*

We write CCS$^1_{vp}$ $\sqsubseteq^{vp}$ CCS$^2_{vp}$ *if there is a subbisimilarity from* CCS$^1_{vp}$ *to* CCS$^2_{vp}$. *The notation $\sqsubseteq^{vp}_{\downarrow}$ is defined accordingly.*

As in the case of CCS, we can define the eight variants of CCS$_{vp}$. For instance CCS$^!_{vp}$ is CCS$_{vp}$ with replication operator rather than fixpoint operator. We shall not investigate in detail the relative expressiveness of the nine variants of value-passing CCS. But it suffices to say that the theory of value-passing CCS parallels that of CCS to a great extent.

The proof of Proposition 1 can be used to show that $\mathrm{CCS}_{vp}^{m} \sqsubseteq^{vp} \mathrm{CCS}_{vp}^{s}$. The negative results $\mathrm{CCS}_{vp}^{m} \not\sqsubseteq_{\downarrow}^{vp} \mathrm{CCS}_{vp}^{s}$ and $\mathrm{CCS}_{vp}^{m\mathrm{def}} \not\sqsubseteq_{\downarrow}^{vp} \mathrm{CCS}_{vp}^{s\mathrm{def}}$ can be proved as in the proof of Proposition 3. Using the proof of $\mathrm{CCS} \not\sqsubseteq_{\downarrow}^{ccs} \mathrm{CCS}^{m\mathrm{def}}$ of Fact 3, we can prove $\mathrm{CCS}_{vp} \not\sqsubseteq_{\downarrow}^{vp} \mathrm{CCS}_{vp}^{m}$ with the help of the process $\mu X.(\overline{a}(0) + \overline{b}(0) \mid X)$. The properties established in Section 2.3.2 do not depend on the particular forms of prefix operations. They are all valid for value-passing CCS. The proof of Lemma 37 can also be transplanted to the value-passing setting. A process in $\mathrm{CCS}_{vp}^{!}$ is structurally bisimilar to a process of the form $\sum_{i\in I} \textbf{if } \varphi_i \textbf{ then } \lambda_i.P_i$ in CCS_{vp}^{gm}, using the expansion law for value-passing CCS. Hence $\mathrm{CCS}_{vp}^{!} \sqsubseteq_{\downarrow}^{vp} \mathrm{CCS}_{vp}^{m} \sqsubseteq_{\downarrow}^{vp} \mathrm{CCS}_{vp}^{!}$. Similarly $\mathrm{CCS}_{vp}^{s} \sqsubseteq_{\downarrow}^{vp} \mathrm{CCS}_{vp}^{s!} \sqsubseteq_{\downarrow}^{vp} \mathrm{CCS}_{vp}^{s}$. The proof of Proposition 10 shows that $\mathrm{CCS}_{vp}^{\mathrm{def}} \sqsubseteq^{vp} \mathrm{CCS}_{vp}^{m\mathrm{def}} \sqsubseteq^{vp} \mathrm{CCS}_{vp}^{s\mathrm{def}}$.

Suppose that P_0 is a CCS_{vp}-process that does not use any operators of the data algebra $\mathfrak{D}$ and that the boolean expressions appeared in P_0 contain only the propositional logic operators and the syntactical equality $\equiv$ judgement. If P_0 induces an infinite action sequence

$$P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \ldots P_i \xrightarrow{\lambda_i} \ldots \tag{43}$$

then using the idea explained in the proof of Theorem 5, one could easily modify the sequence (43) in such a way that a new infinite sequence

$$P_0 \xrightarrow{\lambda_0'} P_1 \xrightarrow{\lambda_1'} \ldots P_i \xrightarrow{\lambda_i'} \ldots \tag{44}$$

is obtained that validates the proposition "$\{v_i \mid \lambda_i' = a_i(v_i)\}$ is finite". Consequently (44) is well quasi-ordered with respect to $\preceq$. This leads immediately to the following proposition.

Proposition 25. CCS_{vp} *is not Turing complete without using the domain knowledge of* $\mathfrak{D}$.

If the data algebra $\mathfrak{D}$ is equipped with enough structure, value-passing CCS would be Turing complete. If this happens, the method we have used to tell apart $\mathrm{CCS}^{\mathrm{def}}$, CCS and $\mathrm{CCS}^{!}$ fails in the value-passing framework. The issue of the relationships among $\mathrm{CCS}_{vp}^{\mathrm{def}}$, CCS_{vp} and $\mathrm{CCS}_{vp}^{!}$ has to be investigated in another occasion.

5.3 Asynchronous Pi

Asynchronous communications are those in which the parties that release messages immediately go ahead without waiting for any acknowledgements from the receiving parties. In the setting of π-calculus the asynchronous output can be defined by the following semantic rules.

$$\frac{}{\overline{a}c.P \xrightarrow{\tau} \overline{a}c \mid P} \qquad \frac{}{\overline{a}c \xrightarrow{\overline{a}c} \mathbf{0}}$$

It is apparent that $\overline{a}c.P$ is bisimilar to $\overline{a}c \mid P$ under this semantics. This fact tells us that if we want to study asynchronous communications in π-calculus, we might as well focus on the following syntax:

$$P \quad := \quad \mathbf{0} \mid n(x).P \mid \overline{n}m \mid P \,|\, P \mid (a)P \mid !P.$$

This is the so-called *asynchronous* π-calculus, often denoted by π^a. A form of restricted output prefixing is definable in the asynchronous calculus using the concurrent composition operator:

$$\overline{a}(c).c(x).P \overset{\text{def}}{=} (c)(\overline{a}c \,|\, c(x).P).$$

The variants of π^a-calculus include the followings:

- π^i is π^a-calculus plus input choice $\sum_{i\in I} a_i(x).P_i$, where I is finite;
- $\pi^a_=$ is π^a-calculus plus match conditional prefix $\varphi a(x).P$;
- $\pi^i_=$ is π^a-calculus plus match conditional choice $\sum_{i\in I} \varphi_i a_i(x).P_i$;
- $\pi^a_{\neq}$ is π^a-calculus plus mismatch conditional prefix $\varphi a(x).P$;
- $\pi^i_{\neq}$ is π^a-calculus plus mismatch conditional choice $\sum_{i\in I} \varphi_i a_i(x).P_i$;
- π^a_c is π^a-calculus plus conditional prefix $\varphi a(x).P$;
- π^i_c is π^a-calculus plus conditional choice $\sum_{i\in I} \varphi_i a_i(x).P_i$.

In the asynchronous calculi, $\overline{a}c$ is a message that has been sent out by some process but has not yet been received by any process. It does not really make sense to place any condition right in front of $\overline{a}c$.

The algebraic theory of π^a-calculus has been studied by several researchers [6, 34, 3, 46, 49, 47]. These researchers have reached to a consensus on how the asynchronous processes can be observed. An asynchronous observer differs in observing power from a synchronous one in that the former cannot really see any input actions performed by observees. Consequently the subbisimilarity relationships between two π^a-variants do not explicitly bisimulate input actions. The abilities to perform input actions are compared by the effects they exert on neighboring processes. A typical equality valid in π^a-calculus is

$$a(x).\overline{a}x \approx \mathbf{0}. \tag{45}$$

Equality (45) is characteristic of the observational theory of the asynchronous calculi. It also underlies most encodings between asynchronous calculi.

The exclusive observability of output actions does not mean that input actions can be ignored. The simple process $a(x)$ for example may consume the process $\overline{a}c$ as it were. So the presence of a fireable input prefix could be detected in a roundabout way. An observational equivalence for asynchronous calculi need to take that into account. In [3] the authors offer basically two approaches to deal with input actions. One is to work out how the input actions are explicitly simulated in a labeled transition semantics. The other is to leave the simulation of input actions implicit but to impose the condition that the observational equivalence is closed under composition. The former is more tractable whereas the latter is more robust. In this paper we opt for the second approach. This is because we will compare π^a-calculus against the synchronous calculus π^s. Only the latter approach provides a uniform tool to carry out the comparison.

Definition 18. *Suppose π_1 and π_2 are two π-variants. An* asynchronous subbisimilarity *from π_1 to π_2 is a total relation $\mathcal{R}$ from $\mathcal{P}_{\pi_1}$ to $\mathcal{P}_{\pi_2}$ such that the following statements are valid.*
(i) If $P_1\mathcal{R}Q_1$ and $P_2\mathcal{R}Q_2$ then $P_1 \,|\, P_2 \;\mathcal{R}\; Q_1 \,|\, Q_2$.
(ii) The following bisimulation property holds whenever $P\mathcal{R}Q$.

- *If $P \xrightarrow{\lambda}_1 P'$ for some $\lambda \in \{\overline{a}c, \overline{a}(c) \mid a, c \in \mathcal{N}\} \cup \{\tau\}$, then $Q \overset{\widehat{\lambda}}{\Longrightarrow}_2 Q'$ for some Q' such that $P'\mathcal{R}Q'$.*
- *If $Q \xrightarrow{\lambda}_2 Q'$ for some $\lambda \in \{\overline{a}c, \overline{a}(c) \mid a, c \in \mathcal{N}\} \cup \{\tau\}$, then $P \overset{\widehat{\lambda}}{\Longrightarrow}_1 P'$ for some P' such that $P'\mathcal{R}Q'$.*

We say that π_1 is asynchronously subbisimilar *to π_2, notation $\pi_1 \sqsubseteq^{asy} \pi_2$, if there is some asynchronous subbisimilarity from the former to the latter. The notations $\sqsubset^{asy}, \sqsubseteq^{asy}_{\downarrow}, \sqsubset^{asy}_{\downarrow}$ are defined accordingly.*

Definition 18 is formulated to reconcile the asynchrony with the synchrony. The following proposition explains.

Proposition 26. *Suppose π^1 and π^2 are two synchronous π-variants. The following statements are valid.*
(i) $\pi^1 \sqsubseteq^{asy} \pi^2$ if and only if $\pi^1 \sqsubseteq^{\pi} \pi^2$.
(ii) $\pi^1 \sqsubseteq^{asy}_{\downarrow} \pi^2$ if and only if $\pi^1 \sqsubseteq^{\pi}_{\downarrow} \pi^2$.

Proof. Suppose π^1 and π^2 are two synchronous π-variants. Let $\mathcal{R}$ be a subbisimilarity from π^1 to π^2 in the sense of Definition 16. We define an infinite sequence of relations from π^1 to π^2 by the following induction.

- $\sqsubseteq_0 \overset{\mathrm{def}}{=} \mathcal{R}$;
- $\sqsubseteq_{i+1} \overset{\mathrm{def}}{=} \left\{ ((a)P_1, (a)P_2), (P_1 \,|\, P_1', P_2 \,|\, P_2') \;\middle|\; \begin{array}{l} P_1 \sqsubseteq_i P_2, \\ P_1' \sqsubseteq_i P_2', \\ \text{and } a \in \mathcal{N} \end{array} \right\}.$

Now let $\sqsubseteq^{\infty}$ be $\bigcup_{i\in\omega} \sqsubseteq_i$. It is a standard exercise to show that $\sqsubseteq^{\infty}$ satisfies the bisimulation property of Definition 18. By construction it is closed under composition. Hence $\pi^1 \sqsubseteq^{asy} \pi^2$. If $\mathcal{R}$ satisfies the codivergence condition, then it follows from the bisimulation property that $\sqsubseteq^{\infty}$ also satisfies the codivergence condition.

Let $\mathcal{A}$ be an asynchronous subbisimilarity from π^1 to π^2. Suppose $P\mathcal{A}Q \xrightarrow{ae} Q'$. For names b, c that appear neither in P nor Q, there must be some process B in π^2 such that $\overline{a}e.\overline{b}c \;\mathcal{A}\; B$. It should not be difficult to see that $B \overset{\overline{a}e}{\Longrightarrow}\overset{\overline{b}c}{\Longrightarrow}\sim \mathbf{0}$.

Consequently $P \,|\, \overline{a}e.\overline{b}c \;\mathcal{A}\; Q \,|\, B \overset{\tau}{\Longrightarrow}\overset{\overline{b}c}{\Longrightarrow}\sim Q' \,|\, \mathbf{0}$. It follows from the bisimulation property that $P \overset{ae}{\Longrightarrow} P'$ and $P' \,|\, \mathbf{0} \;\mathcal{A}\; Q' \,|\, \mathbf{0}$. We conclude that $\sim \mathcal{A} \sim$ is a subbisimilarity from π^1 to π^2 in the sense of Definition 16. Obviously $\sim \mathcal{A} \sim$ satisfies the codivergent condition if and only if $\mathcal{A}$ satisfies the codivergent condition. □

Proposition 26 asserts that asynchronous subbisimilarity provides a uniform criterion for comparing the interactability of both asynchronous π-calculi as well as the synchronous π-calculi. In the rest of this subsection we use the asynchronous subbisimilarity to characterize interactability. The next theorem says that the conditionals also add substantial expressive power in the asynchronous setting.

Proposition 27. *The following statements are valid.*
(i) $\pi^a \sqsubset^{asy} \pi^a_= \sqsubset^{asy} \pi^a_c$; $\pi^a \sqsubset^{asy} \pi^a_{\neq} \sqsubset^{asy} \pi^a_c$.
(ii) $\pi^i \sqsubset^{asy} \pi^i_= \sqsubset^{asy} \pi^i_c$; $\pi^i \sqsubset^{asy} \pi^i_{\neq} \sqsubset^{asy} \pi^i_c$.

Proof. The inclusion maps are all asynchronous subbisimilarities. The processes $a(x).(c)(\overline{c}c \,|\, [x{=}d]c(z).\overline{b}b)$ and $a(x).(c)(\overline{c}c \,|\, [x{\neq}d]c(z).\overline{b}b)$ are in $\pi^a_=$-calculus and $\pi^a_{\neq}$-calculus respectively. They can be used to show that there are no asynchronous subbisimilarities in the opposite directions. Suppose there were an asynchronous subbisimilarity $\mathcal{R}$ from $\pi^a_=$-calculus to π^a-calculus and that $a(x).(c)(\overline{c}c \,|\, [x{=}d]c(z).\overline{b}b) \;\mathcal{R}\; P$ for some P in π^a-calculus. Suppose further that $\overline{a}d\mathcal{R}A$ and $\overline{a}e\mathcal{R}B$. Since $\mathcal{R}$ is closed under composition, one must have

$$\overline{a}d \,|\, a(x).(c)(\overline{c}c \,|\, [x{=}d]c(z).\overline{b}b) \quad \mathcal{R} \quad A \,|\, P, \tag{46}$$

$$\overline{a}e \,|\, a(x).(c)(\overline{c}c \,|\, [x{=}d]c(z).\overline{b}b) \quad \mathcal{R} \quad B \,|\, P. \tag{47}$$

By definition the only observable action A can do is $\overline{a}d$. For the same reason B cannot do any observable actions apart from $\overline{a}e$. It follows from (46) and (47) that

$$P' \overset{\overline{a}e}{\longleftarrow}\Longleftarrow \quad P \quad \overset{\overline{a}d}{\Longrightarrow}\overset{\overline{b}b}{\Longrightarrow} \tag{48}$$

for some P' that cannot do $\overline{b}b$. As in the proof of Proposition 19, the two sequences of actions in (48) contradict each other. □

The relationship between π^a-calculus and π^i-calculus has been studied by Nestmann and Pierce in [46]. They proved a remarkable result that these two calculi are interactively equivalent from the point of view of asynchronous communication.

Fact 5 (Nestmann, Pierce [46]). $\pi^i \sqsubseteq^{asy} \pi^a$.

Proof. The encoding from π^i-calculus to π^a-calculus is homomorphic on all but the input choice operator. Nestmann and Pierce's encoding $[\![\sum_{i\in I} a_i(x).P_i]\!]$ of the input choice $\sum_{i\in I} a_i(x).P_i$ is defined by

$$(c)(\overline{c}(e,f).\overline{e} \,|\, \prod_{i\in I} !a_i(x).c(y,z).Test_i(y,z))$$

where $Test_i(y,z)$ stands for

$$y.(\overline{c}(e,f).\overline{f}\,|\,P_i)\,|\,y.(\overline{c}(e,f).\overline{e}\,|\,\overline{a_i}x)\,|\,z.(\overline{c}(e,f).\overline{f}\,|\,\overline{a_i}x)$$

and $c(y,z).P$, $\overline{c}(e,f).Q$ are defined respectively as follows:

$$\begin{aligned} c(y,z).P &\overset{\text{def}}{=} c(u).\overline{u}(v).v(y).u(z).P, \\ \overline{c}(e,f).Q &\overset{\text{def}}{=} \overline{c}(u).u(v).(e)(f)(\overline{v}e\,|\,\overline{u}f\,|\,Q). \end{aligned}$$

The basic idea is that no input action is prohibited. But once some input has been committed, all the other input actions must be undone. The trick is to use the flag $\overline{c}(e,f).e$ to indicate that no input has been committed and the flag $\overline{c}(e,f).f$ that some input has been committed. The component $y.(\overline{c}(e,f).\overline{e}\,|\,\overline{a_i}x)$ of $Test_i(y,z)$ is indispensable. Without it internal loops would break down. □

The technique used to prove the above fact can be easily adapted to a proof of the following fact.

Fact 6. $\pi^i_= \sqsubseteq^{asy} \pi^a_=$; $\pi^i_{\neq} \sqsubseteq^{asy} \pi^a_{\neq}$; $\pi^i_c \sqsubseteq^{asy} \pi^a_c$.

However the above equivalences fail if computational factor is taken into account. It is conjectured in [46] that there is no termination preserving fully abstract encoding from π^i-calculus to π^a-calculus. The next proposition confirms the conjecture.

Proposition 28. $\pi^a \sqsubset^{asy}_{\downarrow} \pi^i$; $\pi^a_= \sqsubset^{asy}_{\downarrow} \pi^i_=$; $\pi^a_{\neq} \sqsubset^{asy}_{\downarrow} \pi^i_{\neq}$; $\pi^a_c \sqsubset^{asy}_{\downarrow} \pi^i_c$.

Proof. If $\pi^i \sqsubseteq^{asy}_{\downarrow} \pi^a$ were true, there would be an asynchronous subbisimilarity $\mathcal{R}$ from π^i-calculus to π^a-calculus that satisfies the codivergence condition. Then there would be some π^a-process C such that $a(z)+b(z)\ \mathcal{R}\ C$. Since $a(z)+b(z)$ cannot do any τ-action, it follows from the codivergence condition that we may as well assume that the next action of C cannot be τ. Similarly A and B exist such that $\overline{a}a\mathcal{R}A$ and $\overline{b}b\mathcal{R}B$. Again we assume that the next action of A (B) can only be $\overline{a}a$ ($\overline{b}b$). Since $\mathcal{R}$ is closed under composition, one would have that $\overline{a}a\,|\,\overline{b}b\,|\,(a(z)+b(z))\ \mathcal{R}\ A\,|\,B\,|\,C$, from which it follows that C could do immediately an input action on a and an input action on b. But then C could do an input action on a followed by an input action on b, which would lead to a contradiction. □

Nestmann has looked at the encodings from π^s-calculus to π^a-calculus [47]. The adequacy of the encodings is established for some equivalences considerably weaker than asynchronous bisimilarity. As a matter of fact there is little room for improvement.

Proposition 29. $\pi^a \sqsubset^{asy} \pi^-$; $\pi^i \sqsubset^{asy} \pi^s$.

Proof. Suppose A is in π^a-calculus. There is no asynchronous subbisimilarity $\mathcal{R}$ from π^--calculus to π^a-calculus to which $(\overline{a}e.\overline{b}f, A)$ belongs. This can be

$$\begin{tikzcd}
& \pi^s \arrow[dl, shift left, "/" marking] \arrow[dr, shift left, "/" marking] & \\
\pi^i \arrow[ur, tail, shift left] \arrow[dr, shift left] & & \pi^- \arrow[ul, tail, shift left] \arrow[dl, shift left, "/" marking] \\
& \pi^a \arrow[ul, tail, shift left] \arrow[ur, tail, shift left] &
\end{tikzcd}$$

Figure 9: Asynchronous Subbisimilarities

argued as follows: Suppose there were an asynchronous subbisimilarity $\mathcal{R}$ such that $(\overline{a}e.\overline{b}f, A) \in \mathcal{R}$. Now the two consecutive actions $\overline{a}e.\overline{b}f \xrightarrow{\overline{a}e} \overline{b}f \xrightarrow{\overline{b}f} \mathbf{0}$ had to be simulated by $A \Longrightarrow A_1 \xrightarrow{\overline{a}e} A_2 \Longrightarrow A_3 \xrightarrow{\overline{b}f}\Longrightarrow A'$. By the definition of asynchrony, the action $A_1 \xrightarrow{\overline{a}e} A_2$ can be delayed until all the internal actions from A_2 to A_3 have been executed. In other words, there is an A_2' such that $A_1 \Longrightarrow A_2' \xrightarrow{\overline{a}e} A_3$. Using the asynchronous property again, one would have $A_2' \xrightarrow{\overline{b}f}\xrightarrow{\overline{a}e}$. This action sequence is impossible for $\overline{a}e.\overline{b}f$ to simulate.

The argument for the strictness of $\pi^i \sqsubseteq^{asy} \pi^s$ is the same. □

Proposition 29 can be interpreted as saying that the asynchronous variants of π are strictly less expressive than the synchronous counterparts from the viewpoint of interaction.

Finally we state some results that tighten up the remaining loose ends.

Proposition 30. *$\pi^- \not\sqsubseteq^{asy} \pi^i$; $\pi^s \not\sqsubseteq^{asy} \pi^-$; $\pi^i \not\sqsubseteq^{asy}_{\downarrow} \pi^-$.*

Proof. The arguments used in the proof of Proposition 29 are good for $\pi^- \not\sqsubseteq^{asy} \pi^i$. The arguments for Proposition 1 can be modified to prove $\pi^s \not\sqsubseteq^{asy} \pi^-$ and $\pi^i \not\sqsubseteq^{asy}_{\downarrow} \pi^-$. □

We have clarified some issues on the relative expressive power of π^a-variants. The relationships are summarized in Figure 9. In the diagram, the tailed arrow $\rightarrowtail$ is $\sqsubseteq^{asy}_{\downarrow}$; the plain arrow $\rightarrow$ represents $\sqsubseteq^{asy}$; and $\nrightarrow$ indicates $\not\sqsubseteq^{asy}$. The picture of Figure 9 is not compatible with that of Figure 1. For example there is an arrow from π^s-calculus to π^a-calculus in Figure 1. In Figure 9 there is definitely not a plain arrow from π^s-calculus to π^a-calculus for otherwise it would contradict to $\pi^s \nrightarrow \pi^-$. As a matter of fact in Figure 1 there is an arrow from π^s-calculus to π^--calculus by composing two arrows.

5.4 Internal Communication

Sangiorgi's πI-calculus [54] is interesting in that it appears to enjoy much of the expressiveness of π-calculus while retaining the simplicity of the semantics of CCS. The syntax of πI-calculus is as follows:

$$E \ := \ \mathbf{0} \mid X \mid n(x).E \mid \overline{n}(c).E \mid E \,|\, E' \mid (c)E \mid E{+}E' \mid \mu X.E.$$

The calculus has two prefix rules and two communication rules formulated as follows.

$$\frac{}{a(x).E \xrightarrow{ac} E} \qquad \frac{}{\overline{a}(c).E \xrightarrow{\overline{a}(c)} E}$$

$$\frac{E \xrightarrow{ac} E' \quad F \xrightarrow{\overline{a}(c)} F'}{E \,|\, F \xrightarrow{\tau} (c)(E' \,|\, F')} \qquad \frac{E \xrightarrow{\overline{a}(c)} E' \quad F \xrightarrow{ac} F'}{E \,|\, F \xrightarrow{\tau} (c)(E' \,|\, F')}$$

The other rules are the same as those of π-calculus. Without further ado, let's see the definition of bisimulation equivalence.

Definition 19 (Sangiorgi)**.** *A symmetric relation $\mathcal{R}$ on the set of πI processes is a πI-bisimulation if the following condition is met.*

If $P\mathcal{R}Q \xrightarrow{\lambda} Q'$ then $P \overset{\widehat{\lambda}}{\Longrightarrow} P'\mathcal{R}Q'$ for some P'.

The πI-bisimilarity $\approx_I$ is the largest πI-bisimulation. The structural bisimilarity $\sim$ for πI is defined accordingly.

Let πI^{def} be the πI-calculus with parametric definition, $\pi I^!$ be the πI-calculus with replication, and πI^m be the πI-calculus with mixed choice. In the spirit of Definition 19, we may define subbisimilarities, notation $\sqsubseteq^{\pi I}$, and codivergent subbisimilarities, notation $\sqsubseteq^{\pi I}_{\downarrow}$, for πI-variants.

Sangiorgi showed in [54] that πI^{def}-calculus is more expressive than $\pi I^!$-calculus. The interactive behaviour of the following πI^{def}-process

$$D(x) \overset{\mathrm{def}}{=} \overline{x}(c).D(c)$$

is different from any $\pi I^!$-process. For a counter example, the process $D(c)$ can engage in the following infinite sequence of actions

$$D(c) \xrightarrow{\overline{c}(c_1)} \xrightarrow{\overline{c_1}(c_2)} \xrightarrow{\overline{c_2}(c_3)} \ldots$$

which cannot be simulated by any $\pi I^!$-process.

Proposition 31. *The following statements are valid.*
(i) (Sangiorgi) $\pi I^{\mathrm{def}} \not\sqsubseteq^{\pi I} \pi I$, $\pi I \sqsubseteq^{\pi I}_{\downarrow} \pi I^{\mathrm{def}}$.
(ii) $\pi I \sqsubseteq^{\pi I}_{\downarrow} \pi I^m \sqsubseteq^{\pi I}_{\downarrow} \pi I^! \sqsubseteq^{\pi I}_{\downarrow} \pi I$.

Proof. (i) The counter example given above serves as a proof. (ii) The investigations carried out in Section 2.3.2 can be redone for πI-calculus. The conclusion is that πI-calculus, πI^m-calculus and $\pi^!$-calculus are equivalent. The details can be safely omitted. $\square$

6 Effective Subbisimilarity

In Section 2.2.2 we have remarked on the relationship between Church-Turing Thesis and the subbisimilarity approach. In present section we take another look at the relationship by considering Turing completeness.

Turing completeness is a basic criterion for the expressiveness of a process calculus. Intuitively a process calculus is Turing complete if it is capable of encoding all computable functions. According to Church-Turing Thesis, we may investigate Turing completeness property using the following reductional approach.

> A process calculus is Turing complete if there is a translation from a known model of computation to the process calculus.

The above statement immediately suggests a key question. What counts as a legitimate translation from a known model of computation to a process calculus? The answer should be straightforward once the following view is accepted.

> A process calculus is Turing complete if it *is* a model of computation.

Since we think of a τ-action as an atomic computation step, the properties stated in Section 2.2.2 can be repeated in the present setting.

> A translation $\mathfrak{T}$ from a model $(\mathcal{M}, \rightarrow)$ of computation to a process calculus $\mathcal{L}$ is a total relation from the set of the computing objects of $\mathcal{M}$ to $\mathcal{P}_{\mathcal{L}}$ that validates at least the following statements.
>
> 1. If $M\mathfrak{T}P$ and $M \rightarrow M'$ then $P \Longrightarrow P'\mathfrak{T}^{-1}M'$ for some P'.
> 2. If $M\mathfrak{T}P \xrightarrow{\tau} P'$ then $M \rightarrow^* M'\mathfrak{T}P'$ for some M'.
> 3. If $M_0\mathfrak{T}P$ and $M_0 \rightarrow M_1 \rightarrow \ldots M_i \rightarrow \ldots$ is an infinite computation, then $P \overset{\tau}{\Longrightarrow} P'\mathfrak{T}^{-1}M_k$ for some P' and some $k \geq 1$.
> 4. If $M\mathfrak{T}P_0$ and $P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots P_i \xrightarrow{\tau} \ldots$ is an infinite τ-action sequence, then $M \rightarrow^+ M'\mathfrak{T}P_k$ for some M' and some $k \geq 1$.
> 5. Effectiveness Condition: $\mathfrak{T}$ is effective.

Due to space limitation we will refrain from giving a formulation of Turing completeness. But the above properties will serve our purpose.

A number of Turing completeness results have been reported in literature. These results are established with varying strength. Consult [42] for a comparison of different approaches to Turing completeness and [38] for an interpretation of Turing completeness that is very weak from the point of view of interaction. Busi, Gabbrielli and Zavattaro's proof can be easily modified to a total relation from the set of RARM's to $\mathcal{P}_{\mathrm{CCS}^{m\mathrm{def}}}$ that satisfies the above five conditions [7]. Milner has essentially demonstrated in [40] that there is a total relation from the set of the closed λ-terms [5] to $\mathcal{P}_{\pi^!}$ that satisfies the five conditions, where the operational semantics of the λ-terms is given by that of the lazy λ-calculus of Abramsky [1]. The construction is based on an encoding that makes use of a global pointer name e, a countable set $\{f, \ldots\}$ of local pointer names, a countable set $\{c, \ldots\}$ of resource names, a countable set $\{v, \ldots\}$ of name variables, and an injective map from the countable set of the λ-variables $\{x, \ldots\}$ to the set of name variables. For simplicity we assume that the injective map is an identical function. We also assume that there is an effective bijection between $\{x, \ldots\}$ and $\{c, \ldots\}$. The encoding is formally defined as follows:

$$\begin{array}{rcl} [\![x]\!]_p & \overset{\text{def}}{=} & \overline{x}p, \\ [\![\lambda x.M]\!]_p & \overset{\text{def}}{=} & p(x).p(v).[\![M]\!]_v, \\ [\![MN]\!]_p & \overset{\text{def}}{=} & (f)([\![M]\!]_f \,|\, \overline{f}(c).\overline{f}p.!c(v).[\![N]\!]_v). \end{array}$$

A standard interpretation of a closed λ-term M is of the following shape:

$$(c_0 \ldots c_n)(O \,|\, !c_0(v_0).[\![N_0]\!]_{v_0}\{c_1/x_1, \ldots, c_n/x_n\} \,|\, \ldots \,|\, !c_n(v_n).[\![N_n]\!]_{v_n})$$

where O can be in one of the two forms:

- $O \equiv [\![F]\!]_p\{c_0/x_0, \ldots, c_n/x_n\}$ such that $M \equiv F[x_0 := N_0] \ldots [x_n := N_n]$;
- O is the process

$$\begin{array}{rcl} (f_0 \ldots f_m)(c_0)(f^0(v).[\![F]\!]_v & | & \overline{f^0}f^1.!c_0(v_0).[\![N^0]\!]_{v_0} \\ & | & \overline{f^1}(c_1).\overline{f^1}f^2.!c_1(v_1).[\![N^1]\!]_{v_1} \\ & \vdots & \\ & | & \overline{f^m}(c_m).\overline{f^m}p.!c_m(v_m).[\![N^m]\!]_{v_m}) \end{array}$$

  such that $M \equiv (FN^0 \cdots N^m)[x_0 := N_0] \ldots [x_n := N_n]$.

Now suppose $[\![M]\!]_e \overset{\tau}{\Longrightarrow} Q$. Then Q might contain some occurrences of $\mathbf{0}$ that can be safely omitted without affecting the semantics of Q. Using equalities (11) and(12) and the following equality

$$P \,|\, (a)Q \quad = \quad (a)(P \,|\, Q), \text{ if } a \notin gn(P)$$

we can design a converting algorithm $\mathbb{A}$ that transforms Q to a standard interpretation form and then from the standard interpretation form to the corresponding λ-term M. Notice that a standard interpretation form contains enough information from which a λ-term can be constructed. The algorithm $\mathbb{A}$ reports "no" if Q cannot be converted to a standard interpretation form. Now let $\mathfrak{A}$ be the algorithm that, upon receiving a program M, runs the enumeration algorithm for the expressions of $\mathcal{L}$ and checks every expression if it can be converted to M using the algorithm $\mathbb{A}$. If the answer is "yes" then output the expression; otherwise check the next expression. Using this algorithm we can define $\mathfrak{T}$ by the following relation

$$\left\{ (L, P) \,\middle|\, \begin{array}{l} L \text{ is a closed } \lambda \text{ term;} \\ \text{and } P \text{ is in the enumeration set of } \mathfrak{A}(L) \end{array} \right\}.$$

The proofs of Milner [40] can be summarized by saying that the total relation $\mathfrak{T}$ satisfies the five conditions of Turing completeness.

Let's see what is missing in the statements 1-5. A computable function is a procedure that when given an input number, it outputs a number after some calculation. The function may deliver different results in response to different inputs. From the point of view of process theory, an input is picked up at an input channel, whereas an output is delivered at an output channel. Now let F be a process that encodes a computable function. It inputs a number at the channel a and output the result number at the channel b. Let I_a be the process that outputs a number, say i, at channel a. Then the function with the input value i can be coded up by $(a)(I_a \,|\, F_{a,b})$. In similar fashion one could code up a Deterministic Turing Machine together with a value on the input tape by a process of the form $(a)(I_a \,|\, TM_{a,b})$. These observations strongly suggest that a translation from a model of computation to a process calculus that proves Turing completeness of the latter is more or less like a subbisimilarity. The effectiveness of Church-Turing Thesis means that we might as well work with the effective subbisimilarities. Let's see what we can say about the effective subbisimilarities between CCS variants.

Definition 20. *A codivergent subbisimilarity $\mathcal{R}$ from* CCS^1 *to* CCS^2 *is an* effective subbisimilarity *if it satisfies Effectiveness Condition. We will write* $\mathrm{CCS}^1 \sqsubseteq_e^{ccs} \mathrm{CCS}^2$ *if there is an effective subbisimilarity from* CCS^1 *to* CCS^2.

For Definition 20 to make sense, effective subbisimilarities must compose.

Proposition 32. *If* $\mathrm{CCS}^1 \sqsubseteq_e^{ccs} \mathrm{CCS}^2 \sqsubseteq_e^{ccs} \mathrm{CCS}^3$ *then* $\mathrm{CCS}^1 \sqsubseteq_e^{ccs} \mathrm{CCS}^3$.

Proof. The composition of two effective relations is effective. □

In completely the same manner we can effectivise as it were the other codivergent subbisimilarities studied in this paper. The question that has to be asked is this: Do all the positive results, obtained in the previous sections, concerning the codivergent subbisimilarities still hold? Our trust on Church-Turing Thesis would inevitably lead us to believe that the answer is affirmative. In the present paper we have neither the time nor the space to pursue this issue further. It seems necessary though to give a simple example to exhibit the effectiveness of subbisimilarity that is not codivergent.

Proposition 33. *There is a subbisimilarity $\mathcal{R}$ from* CCS^m *to* CCS^s *that satisfies Effectiveness Condition.*

Proof. The encoding $[\![\_]\!]$ from CCS^m to CCS^s translates $\sum_{i=1}^n \lambda_i.P_i$ to

$$\mu Z.(\lambda_1.[\![P_1]\!] + \tau.(\lambda_2.[\![P_2]\!] + \tau.(\ldots \tau.(\lambda_n.[\![P_n]\!] + \tau.Z)\ldots))).$$

In addition to the structures of $\lambda_1.[\![P_1]\!]$, $\lambda_1.[\![P_2]\!]$, ..., $\lambda_n.[\![P_n]\!]$, the encoding $[\![\sum_{i=1}^n \lambda_i.P_i]\!]$ introduces a finite state loop structure. Now expand the decidable relation

$$\mathcal{S} \stackrel{\mathrm{def}}{=} \{(P, [\![P]\!]) \mid P \in \mathcal{P}_{\mathrm{CCS}^m}\} \tag{49}$$

in the following manner: If $P \equiv \sum_{i=1}^{n} \lambda_i.P_i$ then add the following pairs

$$\begin{array}{l}(P, \lambda_1.[\![P_1]\!] + \tau.(\lambda_2.[\![P_2]\!] + \ldots \tau.(\lambda_n.[\![P_n]\!] + \tau.\mu_P)\ldots)),\\ (P, \lambda_2.[\![P_2]\!] + \tau.(\ldots \tau.(\lambda_n.[\![P_n]\!] + \tau.\mu_P)\ldots)),\\ \vdots \\ (P, \lambda_n.[\![P_n]\!] + \tau.\mu_P)\end{array}$$

to the relation, where μ_P is $[\![\sum_{i=1}^{n} \lambda_i.P_i]\!]$. The relation $\mathcal{S}^+$ so generated satisfies Effectiveness Condition. □

The prime motivation for introducing the effective subbisimilarities is to investigate the idea of reduction for calculi of interaction. Such reductions both extend and formalize the reductions among the familiar computational models. The running theme of this paper is to use the effective subbisimilarities to classify the expressiveness of both the interaction models and the computation models in a single framework. For comparison of the interaction models, our slogan is this:

> A model $\mathcal{L}$ is at least as expressive as another model $\mathcal{L}'$ if and only if there is an effective subbisimilarity from $\mathcal{L}$ to $\mathcal{L}'$.

For the computation models we hope to talk about a "subbisimilarity" from one model of computation to another (Church-Turing Thesis) and a "subbisimilarity" from a model of computation to a model of interaction (Turing completeness). No matter how these formalizations turn out to be, the following statement really should be valid.

> Suppose $\mathcal{L}$ is Turing complete. If there is an effective subbisimilarity from $\mathcal{L}$ to $\mathcal{L}'$ then $\mathcal{L}'$ is Turing complete.

So we have an indirect way of proving Turing completeness results.

Our second round discussion has got ourselves into a situation where we are obliged to looking for a universal expressiveness relationship for all models of interaction. This is a much more challenging goal than what has been achieved in this paper. The first author is currently developing a general model theory that addresses this issue [20].

7 Towards a Model Independent Theory

Compared to previous works [48], what is reported in this paper emphasizes more on the unifying framework and model independent approach. We have aimed at constructing a single platform in which we may discuss Church-Turing Thesis, Turing completeness and relative expressiveness of process calculi. This is partly motivated by the belief that, since all of them are essentially about the expressive power of computation/interaction models, they ought to be addressed by similar methodologies. The technicalities and the approaches of this paper have been driven by the mission to search for a model-independent theory of expressiveness. The use of the labeled semantics may have blurred the central theme of this paper, since the reductional semantics has been advocated as an approach that is less dependent on a particular model. While we fully agree that a model-independent theory would be built on the notion of reduction rather than on the interaction labels, we would not play down the significance of the labeled approach. For a set of variants of a particular model, the labeled semantics offers alternative yet more tractable means of study. From the point of view of proof methodology, we could introduce the equivalence relation and the expressive relationships on top of a reduction semantics, and then work out the equivalent characterization in the labeled semantics. What we have done in this paper is to use the latter semantics from the very outset, bypassing the step from the reductional semantics to the equivalent labeled semantics. A positive result of this paper can be easily transferred to a positive result in a reductional framework. Similarly the counter examples given in this paper can be modified to give the same separation results in terms of the reduction semantics.

The subbisimilarities studied in this paper are all between similar models. The relative expressiveness between two quite different models is often far more challenging. One such result is given in [19], where a fully abstract encoding of π-calculus in Calculus of Fair Ambients is defined. This encoding gives rise to a codivergent subbisimilarity in an appropriate sense. Another famous encoding proves that the higher order π-calculus [61, 56] is no more expressive than the (first order) π-calculus. The syntax of the (minimal) higher order π-calculus is defined by the following BNF:

$$E := X \mid a(X).E \mid \overline{a}[E] \mid E \,|\, E' \mid (a)E.$$

Here $a(X).E$ is the higher order input term and $\overline{a}[E]$ is the higher order output term. The reason we have left out the continuation in the output term is to make the point that this particular calculus is nothing but the interactive version of the lazy λ-calculus [1]. The term $(a)(a(X).E \,|\, \overline{a}[F])$ for example is, waving hand, the λ-term $(\lambda x.E)F$. Sangiorgi-Thomsen's encoding makes use of an injective map from the set of the process variables to $\mathcal{N}_v$. The core of the encoding is given by the following definition:

$$\begin{aligned} [\![a(X).E]\!]_{st} &\overset{\text{def}}{=} a(x).[\![E]\!]_{st}, \\ [\![\overline{a}[F]]\!]_{st} &\overset{\text{def}}{=} (c)(\overline{a}c. \,|\, !\overline{c}.[\![F]\!]_{st}). \end{aligned}$$

It is easy to see that the encoding gives rise to a codivergent subbisimilarity in an appropriate sense. For yet another example, let's see how to embed say CCS$^{m\text{def}}$ in π^m-calculus. A subbisimilarity $\mathcal{R}$ from the former to the latter is almost a homomorphic map except that it associates every CCS$^{m\text{def}}$ prefix term $\overline{a}.T$ to every element of the set

$$\{\overline{a}(c).T' \mid F\mathcal{R}T' \text{ and } c \text{ is not a global name in } T'\}$$

and the term $a.T$ to every element of the set

$$\{a(x).T' \mid F\mathcal{R}T' \text{ and } x \text{ is not a free name variable in } T'\}.$$

Under this interpretation a single CCS action is simulated by an infinite number of π-actions. This is a simple example of subbisimilarity whose legitimacy is best justified in a reductional framework. It is also an example that indicates the complexity of the issue. A point worth making is that for π-calculus to be an extension of CCS, which we all believe should be the case, a line between the names and the name variables must be drawn. A consistent understanding of the name-hood is crucial for a model-independent theory of interaction. If the names used by two models have quite different interpretations, it raises the question that if it is really sensible to compare them. This remark suggests the methodological assumption that a model-independent theory of interaction should start with something common to all models in the sense that their semantic interpretations remain the same in all models.

After the submission of the first version of the paper, the first author has been working on a general theory of interaction that intends to provide a uniform treatment of both computation models and interaction models [20]. To give a glimpse of the power of the theory, we point out that the three subbisimilarities indicated in the previous paragraph are all particular instances of the model-independent definition of the subbisimilarities. By applying this general theory, one may streamline part of the process theory. The theory of name-passing calculus for example can be developed as an application of the model-independent theory [23]. The theory of value-passing calculus can be investigated in the same way [21]. The model-independent approach helps address the fundamental issues of interaction models. Some progress has been made along the line. Examples include an interpretation of the full λ-calculus in π-calculus [11] and an extensional Petri Net theory that resolves the problem of compositionality [15]. It is in the light of an emerging picture of a general model-independent theory that the significance of this work can be duly appreciated.

Acknowledgement

The first author is supported by The National 973 Project (2003CB317005) and The National Nature Science Foundation of China (60473006 and 60573002). Both authors would like to extend their appreciations to Xiaojuan Cai, Yuxin Deng, Xiaoju Dong, Caodong He, Ying Jiang, Nicole Kwoh, Xinxin Liu, Huan Long, Jianxin Xue, and Han Zhu for their discussions about this paper. They are indebted to the two anonymous referees for their detailed comments on a previous version of the paper. Their criticisms, questions and suggestions have led to a significant improvement of the paper.

A Proof of Proposition 10

Suppose E is a CCS$^{\text{def}}$ expression and that F, F' are two elements of $\mathfrak{Drv}([\![E]\!])$, where $[\![E]\!]$ is the encoding defined in the proof of Proposition 10. We shall write $F \xrightarrow{\tau_0} F'$ if $F \xrightarrow{\tau} F'$ such that the τ-action is either caused by the explicit τ prefix introduced by the encoding in Figure 2, or by a communication between the local names $e_\top$, $e_\bot$ or h introduced by the encoding in Figure 2. In other words, $F \xrightarrow{\tau_0} F'$ indicates that F internally reduces to F' in one step. We use the standard notation $\xrightarrow{\tau_0}^*$ for the reflexive and transitive closure of $\xrightarrow{\tau_0}$, and $\xrightarrow{\tau_0}^n$ for a sequence of n copies of $\xrightarrow{\tau_0}$'s concatenated one after another. Technically we are using *indexed* labeled transition semantics. But we shall not go into that level of details.

Using the notations introduced above, one could define a relation $\mathcal{R}$ by the following set

$$\left\{ (E_1 \mid \ldots \mid E_n, T_1 \mid \ldots \mid T_n) \;\middle|\; \begin{array}{l} \text{for each } i \in \{1,..,n\},\ (e_\varphi)C^\varphi_{E_i} \xrightarrow{\tau_0}^* T_i, \\ \text{or } (h)V^\varphi_{E_i \mid E} \xrightarrow{\tau_0}^* T_i,\ \text{or } (h)W^\varphi_{E \mid E_i} \xrightarrow{\tau_0}^* T_i \end{array} \right\}.$$

We are about to prove that $\mathcal{R} \sim$ is a subbisimilarity from CCS$^{\text{def}}$ to CCS$^{m\text{def}}$. We need to prove several technical lemmas. First observe that it is impossible for a derivative of C^φ_E to perform a $e_{\overline{\varphi}}$ action or an h $(\overline{h})$ action. Similarly it is impossible for a derivative of $V^\varphi_{E_1 \mid E_2}$ to do a $e_\top$ or $e_\bot$ action. These facts can be stated in the form of the following lemma.

Lemma 46. *The following statements are valid.*
(i) $(e_{\overline{\varphi}})C^\varphi_E \sim (h)C^\varphi_E \sim C^\varphi_E$.
(ii) $(e_\top)V^\varphi_{E_1 \mid E_2} \sim (e_\bot)V^\varphi_{E_1 \mid E_2} \sim V^\varphi_{E_1 \mid E_2}$.
(iii) $(e_\top)W^\varphi_{E_1 \mid E_2} \sim (e_\bot)W^\varphi_{E_1 \mid E_2} \sim W^\varphi_{E_1 \mid E_2}$.

Proof. The equivalences are obvious due to the following facts: In C^φ_E, $V^\varphi_{E_1 \mid E_2}$ and $W^\varphi_{E_1 \mid E_2}$, every occurrence of $C^{\overline{\varphi}}_{E'}$ or $e_{\overline{\varphi}}$ is restricted by $(e_{\overline{\varphi}})$; and in C^φ_E, every occurrence of $V^\varphi_{E_1 \mid E_2}$ or $W^\varphi_{E_1 \mid E_2}$ is restricted by (h). $\square$

The next lemma is about similar properties.

Lemma 47. *If either* $(e_\varphi)C^\varphi_E \xrightarrow{\tau_0}^* T$, *or* $(h)V^\varphi_{E_1 \mid E_2} \xrightarrow{\tau_0}^* T$ *or* $(h)W^\varphi_{E_1 \mid E_2} \xrightarrow{\tau_0}^* T$, *then* $(e_\top)T \sim (e_\bot)T \sim (h)T \sim T$.

Proof. This is straightforward from the definition. Notice that every occurrence of $C^\varphi_{E_1 \mid E_2}$ in $V^\varphi_{E_1 \mid E_2}$ is prefixed by h. $\square$

It should be clear from the definition of the encoding that once C^φ_E, or $V^\varphi_{E_1 \mid E_2}$ or $W^\varphi_{E_1 \mid E_2}$ departs from an internal looping, it throws away all global occurrences of $e_\top, e_\bot, h$.

Lemma 48. *Let F be either C^φ_E, or $V^\varphi_{E_1 \mid E_2}$ or $W^\varphi_{E_1 \mid E_2}$. If $F \xrightarrow{\tau_0}^* \xrightarrow{\lambda} F'$ for some $\lambda \notin \{e_\top, e_\bot, h, \tau_0\}$, then none of $e_\top, e_\bot, h$ occurs as global names in F'.*

The above three lemmas are about what $C^\varphi_E, V^\varphi_{E_1 \mid E_2}, W^\varphi_{E_1 \mid E_2}$ cannot do. The next simple lemma is about what they can do. For the sake of stating the following lemmas, we call a process *inactive* if it is composed of localization, composition and $\mathbf{0}$. Clearly an inactive process is structurally bisimilar to $\mathbf{0}$.

An *inactive context* $I[\_]$ is an inactive process with a hole. If $I[\_]$ is an inactive context then $I[E]$ is structurally bisimilar to E whenever none of the global names of E gets localized by $I[\_]$.

Lemma 49. *The following statements are valid.*
(i) $C^{\varphi}_{E} \xrightarrow{\overline{e_\varphi}} I$ *for some inactive process* I.
(ii) $V^{\varphi}_{E_1 \mid E_2} \xrightarrow{h} C^{\varphi}_{E_1 \mid E_2}$.
(iii) $W^{\varphi}_{E_1 \mid E_2} \xrightarrow{\overline{h}} \mathbf{0}$.

The definition of C^{φ}_{E} makes sure that after it performs an $\overline{e_\varphi}$ action after some internal looping, it must turn into some process equivalent to $\mathbf{0}$. Similar properties hold of $V^{\varphi}_{E_1 \mid E_2}$ and $W^{\varphi}_{E_1 \mid E_2}$. Hence the following lemma.

Lemma 50. *The following statements are valid.*
(i) If $C^{\varphi}_{E} \xrightarrow{\tau_0}{}^{*} F \xrightarrow{\overline{e_\varphi}}$ *then* $F \equiv I[C^{\varphi}_{E}]$ *for some inactive context* $I[\_]$.
(ii) If $V^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{*} F \xrightarrow{h}$ *then* $F \equiv I[V^{\varphi}_{E_1 \mid E_2}]$ *for some inactive context* $I[\_]$.
(iii) If $W^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{*} F \xrightarrow{\overline{h}}$ *then* $F \equiv I[W^{\varphi}_{E_1 \mid E_2}]$ *for some inactive context* $I[\_]$.

Proof. Let F_0 be either C^{φ}_{E}, or $V^{\varphi}_{E_1 \mid E_2}$ or $W^{\varphi}_{E_1 \mid E_2}$. We prove the lemma by mutual induction on n, the length of $F_0 \xrightarrow{\tau_0}{}^{n} F$. The case $n = 0$ is obvious. Suppose that the lemma is true for all $n < k$, we show that it is true for $n = k$.

- $F_0 \equiv C^{\varphi}_{\mathbf{0}}$ or $F_0 \equiv C^{\varphi}_{\lambda.E_1}$: These cases are impossible.
- $F_0 \equiv C^{\varphi}_{(c)E'}$: In this case we continue the induction with $C^{\varphi}_{E'}$.
- $F_0 \equiv C^{\varphi}_{E_1+E_2}$: Suppose the first step looping is

$$C^{\varphi}_{E_1+E_2} \xrightarrow{\tau_0} (e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}).$$

Since $C^{\overline{\varphi}}_{E_1} \sim (e_\varphi)C^{\overline{\varphi}}_{E_1}$ by Lemma 46, it must be the case that $C^{\overline{\varphi}}_{E_1} \xrightarrow{\tau_0}{}^{*} \not\xrightarrow{\overline{e_\varphi}}$, meaning that it is impossible for $C^{\overline{\varphi}}_{E_1}$ to do a $\overline{e_\varphi}$ action after some internal looping. So it has to be that $C^{\overline{\varphi}}_{E_1} \xrightarrow{\tau_0}{}^{n'} G \xrightarrow{\overline{e_{\overline{\varphi}}}}$ for some G and some n_1 less than k. By induction $G \equiv I_1[C^{\overline{\varphi}}_{E_1}]$ for some inactive context $I_1[\_]$. Consequently

$$\begin{array}{rcl} F_0 & \xrightarrow{\tau_0}{}^{(n'+1)} & (e_{\overline{\varphi}})(I_1[C^{\overline{\varphi}}_{E_1}] \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}) \\ & \xrightarrow{\tau_0} & (e_{\overline{\varphi}})(I_1[R] \mid C^{\varphi}_{E_1+E_2}) \\ & \xrightarrow{\tau_0}{}^{(k-n'-2)} & F \end{array}$$

for some inactive process R. It is obvious that $F \equiv (e_{\overline{\varphi}})(I_1[R] \mid F')$ for some F' such that $C^{\varphi}_{E_1+E_2} \xrightarrow{\tau_0}{}^{(k-n'-2)} F' \xrightarrow{\overline{e_\varphi}}$. By induction hypothesis, $F' \equiv I_2[C^{\varphi}_{E_1+E_2}]$ for some inactive context $I_2[\_]$. Consequently $F \equiv (e_{\overline{\varphi}})(I_1[R] \mid I_2[C^{\varphi}_{E_1+E_2}])$.

If $C^{\varphi}_{E_1+E_2} \xrightarrow{\tau_0} (e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_2} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2})$ is the first internal looping step, the argument is the same.

- $F_0 \equiv C^{\varphi}_{E_1 \mid E_2}$: The first internal looping step must be

$$C^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0} (h)(V^{\varphi}_{E_1 \mid E_2} \mid W^{\varphi}_{E_1 \mid E_2}).$$

  By Lemma 46, neither $V^{\varphi}_{E_1 \mid E_2}$ nor $W^{\varphi}_{E_1 \mid E_2}$ can ever do a $\overline{e_{\varphi}}$ action. No matter how $V^{\varphi}_{E_1 \mid E_2}$ and $W^{\varphi}_{E_1 \mid E_2}$ loop, they must synchronize at h at some point before reaching F. In other words,

$$V^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{n_1} G_1 \xrightarrow{h}$$

  and

$$W^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{n_2} G_2 \xrightarrow{\overline{h}}$$

  for some G_1, G_2 and some n_1, n_2 less than k. By induction $G_1 \equiv I_1[V^{\varphi}_{E_1 \mid E_2}]$ and $G_2 \equiv I_2[W^{\varphi}_{E_1 \mid E_2}]$ for some inactive contexts $I_1[\_], I_2[\_]$. Consequently

$$C^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{n_1+n_2+2} (h)(I_1[C^{\varphi}_{E_1 \mid E_2}] \mid I_2[\mathbf{0}]).$$

  As in the previous case, we can now proceed by applying the induction hypothesis to the rest of the internal looping.

- $F_0 \equiv V^{\varphi}_{E_1 \mid E_2}$ or $F_0 \equiv W^{\varphi}_{E_1 \mid E_2}$: The proof is similar.

We are done. □

Another important property of C^{φ}_{E} is that whatever internal looping it has engaged, it retains the possibility to go back to C^{φ}_{E}. Similar property holds of $V^{\varphi}_{E_1 \mid E_2}$ and $W^{\varphi}_{E_1 \mid E_2}$.

Lemma 51. *Let F_0 be either C^{φ}_{E}, or $V^{\varphi}_{E_1 \mid E_2}$ or $W^{\varphi}_{E_1 \mid E_2}$. If $F_0 \xrightarrow{\tau_0}{}^{*} F$ then $F \xrightarrow{\tau_0}{}^{*} A \sim F_0$ for some A.*

Proof. We prove the lemma by induction on n, the length of the internal looping $F_0 \xrightarrow{\tau_0}{}^{n} F$. The case of $n = 0$ is trivial. Suppose the lemma is true for all $n < k$, we show that it is true for $n = k$ as well.

- $F_0 \equiv C^{\varphi}_{\mathbf{0}}$ or $F_0 \equiv C^{\varphi}_{\lambda.E}$: In this case there is nothing to prove.

- $F_0 \equiv C^{\varphi}_{(c)E_1}$: In this case we continue with $C^{\varphi}_{E_1}$.

- $F_0 \equiv C^{\varphi}_{E_1+E_2}$: Suppose the first internal looping step is

$$C^{\varphi}_{E_1+E_2} \xrightarrow{\tau_0} (e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}).$$

  There are two subcases.

  - If $C^{\overline{\varphi}}_{E_1} \xrightarrow{\tau_0}{}^{j} \xrightarrow{\overline{e_\varphi}} G$ for some $G \sim \mathbf{0}$ and some $j < k$, then

$$F_0 \xrightarrow{\tau_0}{}^{j+2} (e_{\overline{\varphi}})(G \mid C^{\varphi}_{E_1+E_2}) \xrightarrow{\tau_0}{}^{n-j-2} (e_{\overline{\varphi}})(G \mid F') \equiv F$$

    for some F' such that $C^{\varphi}_{E_1+E_2} \xrightarrow{\tau_0}{}^{n-j-2} F'$. By induction hypothesis, $F' \xrightarrow{\tau_0}{}^{*} A' \sim C^{\varphi}_{E_1+E_2}$ for some A'. Thus $F \xrightarrow{\tau_0}{}^{*} (e_{\overline{\varphi}})(G \mid A') \sim F_0$.

  - If $C^{\overline{\varphi}}_{E_1} \xrightarrow{\tau_0}{}^{n-1} H$ for some H such that $(e_{\overline{\varphi}})(H \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}) \equiv F$, then by induction $H' \xrightarrow{\tau_0}{}^{*} A' \sim C^{\overline{\varphi}}_{E_1}$ for some A'. By Lemma 49, some A'' exists such that $A' \xrightarrow{\overline{e_{\overline{\varphi}}}} A'' \sim \mathbf{0}$. But then

$$F \xrightarrow{\tau_0}{}^{*} (e_{\overline{\varphi}})(A' \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}) \xrightarrow{\tau_0} (e_{\overline{\varphi}})(A'' \mid C^{\varphi}_{E_1+E_2}) \sim F_0.$$

- $F_0 \equiv C^{\varphi}_{E_1 \mid E_2}$: Obviously $C^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0} (h)(V^{\varphi}_{E_1 \mid E_2} \mid W^{\varphi}_{E_1 \mid E_2})$. Suppose $V^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{n_1} G$ and $W^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{n_2} H$ are maximal internal looping in $F_0 \xrightarrow{\tau_0}{}^{n} F$. There are two subcases.

  - $n_1 + n_2 = n - 1$. By induction hypothesis, $G \xrightarrow{\tau_0}{}^{*} A' \sim V^{\varphi}_{E_1 \mid E_2}$ and $H \xrightarrow{\tau_0}{}^{*} B' \sim W^{\varphi}_{E_1 \mid E_2}$ for some A', B'. It follows from Lemma 49 that $A' \xrightarrow{h} A'' \sim C^{\varphi}_{E_1 \mid E_2}$ and $B' \xrightarrow{\overline{h}} B'' \sim \mathbf{0}$ for some A'', B''. Consequently

$$F \xrightarrow{\tau_0}{}^{*} (h)(A' \mid B') \xrightarrow{\tau_0} (h)(A'' \mid B'') \sim C^{\varphi}_{E_1 \mid E_2}.$$

  - $n_1 + n_2 < n - 1$. The next internal looping is an interaction between G and H at u. By Lemma 50, there are inactive contexts $I_1[\_], I_1[\_]$ such that

$$\begin{array}{rcl} G & \equiv & I_1[V^{\varphi}_{E_1 \mid E_2}], \\ H & \equiv & I_2[W^{\varphi}_{E_1 \mid E_2}]. \end{array}$$

    It follows that

$$\begin{array}{rcl} C^{\varphi}_{E_1 \mid E_2} & \xrightarrow{\tau_0} & (h)(V^{\varphi}_{E_1 \mid E_2} \mid W^{\varphi}_{E_1 \mid E_2}) \\ & \xrightarrow{\tau_0}{}^{*} & (h)(I_1[V^{\varphi}_{E_1 \mid E_2}] \mid I_2[W^{\varphi}_{E_1 \mid E_2}]) \\ & \xrightarrow{\tau_0}{}^{*} & (h)(I_1[C^{\varphi}_{E_1 \mid E_2}] \mid I_2[\mathbf{0}]). \end{array}$$

    Clearly $F \equiv (h)(I_1[F'] \mid I_2[\mathbf{0}])$ and $C^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{n'} F'$ for some $n' < n$. By induction hypothesis, $F' \xrightarrow{\tau_0}{}^{*} A' \sim C^{\varphi}_{E_1 \mid E_2}$ for some A'. So $F \xrightarrow{\tau_0}{}^{*} (h)(I_1[A'] \mid I_2[\mathbf{0}]) \sim C^{\varphi}_{E_1 \mid E_2}$.

- $F_0 \equiv V^{\varphi}_{E_1 \mid E_2}$ or $F_0 \equiv W^{\varphi}_{E_1 \mid E_2}$: The proof is similar.

We are done. $\square$

The next lemma shows that the encoding defined in Figure 2 is faithful with respect to the operational semantics.

Lemma 52. *The following statements are valid whenever* $E \xrightarrow{\lambda} E'$.
(i) $C^{\varphi}_{E} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T'$ *for some* T' *such that* $E'\mathcal{R} \sim T'$.
(ii) $V^{\varphi}_{E \mid F} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T'$ *for some* T' *such that* $E'\mathcal{R} \sim T'$.
(iii) $W^{\varphi}_{E \mid F} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T'$ *for some* T' *such that* $E'\mathcal{R} \sim T'$.

Proof. The following is a mutual induction on the height of the inference tree of $E \xrightarrow{\lambda} E'$:

- $\lambda.E \xrightarrow{\lambda} E$: Then $C^{\varphi}_{\lambda.E} \xrightarrow{\lambda} (e_{\varphi})C^{\varphi}_{E}$. It is obvious that $E\mathcal{R}(e_{\varphi})C^{\varphi}_{E}$.

- $E \xrightarrow{\lambda} E' \Rightarrow (c)E \xrightarrow{\lambda} (c)E'$: By induction we have $C^{\varphi}_{E} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T''$ and $E'\mathcal{R} \sim T''$ for some T''. Since $C^{\varphi}_{(c)E} \equiv (c)C^{\varphi}_{E}$, we must have $C^{\varphi}_{(c)E} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} (c)T''$. It is easy to see that $(c)E'\mathcal{R} \sim (c)T''$.

- $E_1 \xrightarrow{\lambda} E_1' \Rightarrow E_1 + E_2 \xrightarrow{\lambda} E_1'$: By induction we have $C^{\varphi}_{E_1} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T_1'$ and $E_1'\mathcal{R} \sim T_1'$ for some T_1'. But then $C^{\varphi}_{E_1+E_2} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} (e_{\overline{\varphi}})(T_1' \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}) \sim T_1'$ by Lemma 48.

- $E_1 \xrightarrow{\lambda} E_1' \Rightarrow E_1 \mid E_2 \xrightarrow{\lambda} E_1' \mid E_2$: By induction we have that $C^{\varphi}_{E_1} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T_1'$ and $E_1'\mathcal{R} \sim T_1'$ for some T_1'. It follows from Lemma 48 that

$$V^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} (e_{\overline{\varphi}})(T_1' \mid e_{\overline{\varphi}}.V^{\varphi}_{E_1 \mid E_2}) \sim T_1'.$$

  So $C^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0} (h)(V^{\varphi}_{E_1 \mid E_2} \mid W^{\varphi}_{E_1 \mid E_2}) \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} \sim (h)(T_1' \mid W^{\varphi}_{E_1 \mid E_2}) \sim T_1' \mid (h)W^{\varphi}_{E_1 \mid E_2}$ by Lemma 48. It is clear that $E_1' \mid E_2\mathcal{R} \sim T_1' \mid (h)W^{\varphi}_{E_1 \mid E_2}$.

- $E_1 \xrightarrow{a} E_1' \wedge E_2 \xrightarrow{\overline{a}} E_2' \Rightarrow E_1 \mid E_2 \xrightarrow{\tau} E_1' \mid E_2'$: By induction we have $V^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{*} \xrightarrow{a} T_1'$ and $E_1'\mathcal{R} \sim T_1'$ for some T_1', and $W^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\overline{a}} T_2'$ and $E_2'\mathcal{R} \sim T_2'$ for some T_2'. Consequently

$$C^{\varphi}_{E_1 \mid E_2} \xrightarrow{\tau_0} (h)(V^{\varphi}_{E_1 \mid E_2} \mid W^{\varphi}_{E_1 \mid E_2}) \xrightarrow{\tau_0}{}^{*} \xrightarrow{\tau} (h)(T_1' \mid T_2') \sim T_1' \mid T_2'$$

  by Lemma 48. Obviously $E_1' \mid E_2'\mathcal{R} \sim T_1' \mid T_2'$.

- $V^{\varphi}_{E \mid F} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T'$: By Lemma 50, some inactive context $I[\_]$ and some process T_1' exist such that

$$\begin{aligned} V^{\varphi}_{E \mid F} & \xrightarrow{\tau_0}{}^{*} & I[(e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E} \mid e_{\overline{\varphi}}.V^{\varphi}_{E \mid F})] \\ & \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} & I[(e_{\overline{\varphi}})(T_1' \mid e_{\overline{\varphi}}.V^{\varphi}_{E \mid F})] \equiv T' \end{aligned}$$

and $C^{\overline{\varphi}}_{E} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T'_1$. By induction, $E'\mathcal{R} \sim T'_1 \sim T'$.

- $W^{\overline{\varphi}}_{F\,|\,E} \xrightarrow{\tau_0}{}^{*} \xrightarrow{\lambda} T'$: The proof is similar.

This completes the induction. □

Actually the encoding defined in Figure 2 is more than faithful. It reflects the operational semantics in the sense of the following lemma.

Lemma 53. *The following statements are valid if* $\lambda \notin \{\overline{e_\varphi}, u, \overline{u}, \tau_0\}$.
(i) If $C^{\varphi}_{E} \xrightarrow{\tau_0}{}^{*} T \xrightarrow{\lambda} T'$, *then* $E \xrightarrow{\lambda} E'$ *and* $E'\mathcal{R} \sim T'$ *for some* E'.
(ii) If $V^{\varphi}_{E_1\,|\,E_2} \xrightarrow{\tau_0}{}^{*} T \xrightarrow{\lambda} T'$, *then* $E_1 \xrightarrow{\lambda} E'$ *and* $E'\mathcal{R} \sim T'$ *for some* E'.
(iii) If $W^{\varphi}_{E_1\,|\,E_2} \xrightarrow{\tau_0}{}^{*} T \xrightarrow{\lambda} T'$, *then* $E_2 \xrightarrow{\lambda} E'$ *and* $E'\mathcal{R} \sim T'$ *for some* E'.

Proof. Let F_0 be either C^{φ}_{E}, or $V^{\varphi}_{E_1\,|\,E_2}$ or $W^{\varphi}_{E_1\,|\,E_2}$. We prove the lemma by induction on n, the length of $F_0 \xrightarrow{\tau_0}{}^{*} T$. When $n = 0$, the only possibility is $F_0 \equiv C^{\varphi}_{\lambda.E}$. In this case $C^{\varphi}_{\lambda.E} \xrightarrow{\lambda} (e_\varphi)C^{\varphi}_{E}$. Clearly $\lambda.E \xrightarrow{\lambda} E$ and $E\,\mathcal{R}\,(e_\varphi)C^{\varphi}_{E}$ by the definition of $\mathcal{R}$. Suppose that the lemma is true for $n < k$, we prove that it is also true for $n = k$. We carry out an induction as follows:

- $F_0 \equiv C^{\varphi}_{(c)E_1}$: In this case we continue the structural induction on $C^{\varphi}_{E_1}$.
- $F_0 \equiv C^{\varphi}_{E_1+E_2}$: Assume that the first internal looping step is
$$C^{\varphi}_{E_1+E_2} \xrightarrow{\tau_0} (e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2}).$$
There are two possibilities:
  - $C^{\overline{\varphi}}_{E_1} \xrightarrow{\tau_0}{}^{n-1} T_1 \xrightarrow{\lambda} T'_1$. By induction hypothesis, there exists some E'_1 such that $E_1 \xrightarrow{\lambda} E'_1$ and $E'_1\mathcal{R} \sim T'_1$. Hence $E_1{+}E_2 \xrightarrow{\lambda} E'_1\mathcal{R} \sim T' \equiv (e_{\overline{\varphi}})(T'_1 \mid e_{\overline{\varphi}}.C^{\varphi}_{E_1+E_2})$ by Lemma 48.
  - $C^{\overline{\varphi}}_{E_1} \xrightarrow{\tau_0}{}^{n_1} T_1 \xrightarrow{\overline{e_{\overline{\varphi}}}} T'_1$ for some n_1 less than k. By Lemma 50, T'_1 is an inactive process. It follows that $C^{\varphi}_{E_1+E_2} \xrightarrow{\tau_0}{}^{n_2} T_2 \xrightarrow{\lambda} T'_2$ for some T_2, T'_2 and some n_2 less than k. By induction hypothesis, $E_1{+}E_2 \xrightarrow{\lambda} E'_1\mathcal{R} \sim T'_2 \equiv (e_{\overline{\varphi}})(T'_1 \mid T'_2)$.
- $F_0 \equiv C^{\varphi}_{E_1\,|\,E_2}$: The first internal looping step is
$$C^{\varphi}_{E_1\,|\,E_2} \xrightarrow{\tau_0} (h)(V^{\varphi}_{E_1\,|\,E_2} \mid W^{\varphi}_{E_1\,|\,E_2}).$$
There are several subcases.
  - $V^{\varphi}_{E_1\,|\,E_2} \xrightarrow{\tau_0}{}^{n-1} T_1 \xrightarrow{\lambda} T'_1$. By induction hypothesis, $E_1 \xrightarrow{\lambda} E'_1\mathcal{R} \sim T'_1$ for some E'_1. It follows that
$$E_1 \,|\, E_2 \xrightarrow{\lambda} E'_1 \,|\, E_2\mathcal{R} \sim T'_1 \,|\, (h)W^{\varphi}_{E_1\,|\,E_2} \sim (h)(T'_1 \,|\, W^{\varphi}_{E_1\,|\,E_2}).$$

- $W^{\varphi}_{E_1 \,|\, E_2} \xrightarrow{\tau_0}{}^{n-1} T_2 \xrightarrow{\lambda} T_2'$. This is symmetric to the previous case.
- $\lambda = \tau$, and the tau action is caused by $V^{\varphi}_{E_1 \,|\, E_2} \xrightarrow{\tau_0}{}^{n_1} T_1 \xrightarrow{a} T_1'$ and $W^{\varphi}_{E_1 \,|\, E_2} \xrightarrow{\tau_0}{}^{n_2} T_2 \xrightarrow{\overline{a}} T_2'$ with $n_1 + n_2 = n - 1$. By induction hypothesis,
$$E_1 \xrightarrow{a} E_1' \mathcal{R} \sim T_1'$$
and
$$E_2 \xrightarrow{\overline{a}} E_2' \mathcal{R} \sim T_2'.$$
Hence $E_1 \,|\, E_2 \xrightarrow{\tau} E_1' \,|\, E_2' \mathcal{R} \sim T_1' \,|\, T_2' \sim (h)(T_1' \,|\, T_2')$.
- $V^{\varphi}_{E_1 \,|\, E_2} \xrightarrow{\tau_0}{}^{n_1} T_1 \xrightarrow{h} T_1'$ and $W^{\varphi}_{E_1 \,|\, E_2} \xrightarrow{\tau_0}{}^{n_2} T_2 \xrightarrow{\overline{h}} T_2'$ with $n_1 + n_2 < k$. By Lemma 50, $T_1' \equiv I_1[C^{\varphi}_{E_1 \,|\, E_2}]$ and $T_2' \equiv I_2[\mathbf{0}]$ for some inactive contexts $I_1[\_], I_2[\_]$. Therefore $C^{\varphi}_{E_1 \,|\, E_2} \xrightarrow{\tau_0}{}^{*} (h)(I_1[C^{\varphi}_{E_1 \,|\, E_2}] \,|\, I_2[\mathbf{0}])$. So we may continue with the induction.

- $F_0 \equiv V^{\varphi}_{E_1 \,|\, E_2}$. The first internal looping step must be
$$V^{\varphi}_{E_1 \,|\, E_2} \xrightarrow{\tau_0} (e_{\overline{\varphi}})(C^{\overline{\varphi}}_{E_1} \,|\, e_{\overline{\varphi}}.V^{\varphi}_{E_1|E_2}).$$
There are two subcase:
  - $C^{\overline{\varphi}}_{E_1} \xrightarrow{\tau_0}{}^{n-1} T_1 \xrightarrow{\lambda} T_1'$. In this case we resort to induction hypothesis.
  - $C^{\overline{\varphi}}_{E_1} \xrightarrow{\tau_0}{}^{n_2} T_2 \xrightarrow{\overline{e_{\overline{\varphi}}}} T_2'$ for some n_2 less than k. By Lemma 50, T_2' is an inactive process. Therefore
$$\begin{aligned} V^{\varphi}_{E_1 \,|\, E_2} & \xrightarrow{\tau_0}{}^{n_2+2} && (e_{\overline{\varphi}})(T_2' \,|\, V^{\varphi}_{E_1|E_2}) \\ & \xrightarrow{\tau_0}{}^{n-n_2-2} && (e_{\overline{\varphi}})(T_2' \,|\, T_2'') \end{aligned}$$
for some T_2''. Therefore we may proceed with $V^{\varphi}_{E_1|E_2} \xrightarrow{\tau_0}{}^{n-n_2-2} T_2''$.
- $F_0 \equiv W^{\varphi}_{E_1 \,|\, E_2}$. This case is symmetric to the previous one.

We are done. □

The faithful and reflection properties allow us to prove the claim we made in the beginning of the section.

Proposition 34. $\mathcal{R} \sim$ *is a subbisimilarity.*

Proof. Assume $E_1 \,|\, E_2 \,|\, \cdots \,|\, E_k \;\mathcal{R}\; T_1 \,|\, T_2 \,|\, \cdots \,|\, T_k \sim T$.

- Suppose $E_1 \,|\, E_2 \,|\, \cdots \,|\, E_k \xrightarrow{\lambda} E'$ for some E'. Without loss of generality there are basically two cases: either
$$E_1 \,|\, E_2 \,|\, \cdots \,|\, E_k \xrightarrow{\lambda} E_1' \,|\, E_2 \,|\, ... \,|\, E_k$$

is caused by $E_1 \xrightarrow{\lambda} E_1'$, or

$$E_1 \,|\, E_2 \,|\, \cdots \,|\, E_k \xrightarrow{\tau} E_1' \,|\, E_2' \,|\, ... \,|\, E_k$$

is caused by an interaction between E_1 and E_2. Consider the first case. We have by definition that $F_1 \xrightarrow{\tau_0}{}^{*} T_1$, where F_1 is either $(e_\varphi)C^{\varphi}_{E_1}$, or $(h)V^{\varphi}_{E_1\,|\,E}$ or $(h)W^{\varphi}_{E\,|\,E_1}$ for some E. By Lemma 51, $T_1 \xrightarrow{\tau_0}{}^{*}\sim F_1$. By Lemma 52, we have that $F_1 \xrightarrow{\tau_0}{}^{*}\xrightarrow{\lambda} T_1'$ and $E_1'\mathcal{R} \sim T_1'$ for some T_1'. Thus $T \Longrightarrow\xrightarrow{\lambda} T'$ for some T' such that

$$E_1' \,|\, E_2 \,|\, \cdots \,|\, E_k \,\mathcal{R} \sim T_1' \,|\, T_2 \,|\, \cdots \,|\, T_k \sim T'.$$

For the second case, suppose $E_1 \xrightarrow{a} E_1'$ and $E_2 \xrightarrow{\overline{a}} E_2'$. By Lemma 51 and Lemma 52, there exist some T_1', T_2' and some $l_1, l_2 \geq 0$ such that

$$T_1 \xrightarrow{\tau_0}{}^{*}\sim F_1 \xrightarrow{\tau_0}{}^{*}\xrightarrow{a} T_1' \sim \mathcal{R}^{-1}E_1'$$

and

$$T_2 \xrightarrow{\tau_0}{}^{*}\sim F_2 \xrightarrow{\tau_0}{}^{*}\xrightarrow{\overline{a}} T_2' \sim \mathcal{R}^{-1}E_2'$$

where F_i, for $i \in \{1,2\}$, is either $(e_\varphi)C^{\varphi}_{E_i}$, or $(h)V^{\varphi}_{E_i\,|\,E}$ or $(h)W^{\varphi}_{E\,|\,E_i}$. It should be then clear that $T \overset{\tau}{\Longrightarrow}\sim T_1' \,|\, T_2' \,|\, ... \,|\, T_k \sim \mathcal{R}^{-1}E_1' \,|\, E_2' \,|\, ... \,|\, E_k$.

- If $T \xrightarrow{\lambda} T'$ then by the definition of structural bisimilarity, we have without loss of generality that either

  $$T_1 \,|\, T_2 \,|\, \cdots \,|\, T_k \xrightarrow{\lambda} T_1' \,|\, T_2 \,|\, \cdots \,|\, T_k \sim T'$$

  is caused by $T_1 \xrightarrow{\lambda} T_1'$, or

  $$T_1 \,|\, T_2 \,|\, \cdots \,|\, T_k \xrightarrow{\tau} T_1' \,|\, T_2' \,|\, \cdots \,|\, T_k \sim T'$$

  is caused by $T_1 \xrightarrow{a} T_1'$ and $T_2 \xrightarrow{\overline{a}} T_2'$, or

  $$T_1 \,|\, T_2 \,|\, \cdots \,|\, T_k \xrightarrow{\tau_0} T_1' \,|\, T_2 \,|\, \cdots \,|\, T_k \sim T'$$

  is caused by $T_1 \xrightarrow{\tau_0} T_1'$. For the first case we have by definition that $F_1 \xrightarrow{\tau_0}{}^{*} T_1 \xrightarrow{\lambda} T_1'$ where F_1 is either $(e_\varphi)C^{\varphi}_{E_1}$, or $(h)V^{\varphi}_{E_1\,|\,E}$ or $(h)W^{\varphi}_{E\,|\,E_1}$ for some E. According to Lemma 53, there exists some E_1' such that $E_1 \xrightarrow{\lambda} E_1'\mathcal{R} \sim T_1'$. Thus $E_1' \,|\, E_2 \,|\, \cdots \,|\, E_k \,\mathcal{R} \sim T_1' \,|\, T_2 \,|\, \cdots \,|\, T_k \sim T'$. For the second case, we have $F_1 \xrightarrow{\tau_0}{}^{*} T_1 \xrightarrow{a} T_1'$ and $F_2 \xrightarrow{\tau_0}{}^{*} T_2 \xrightarrow{\overline{a}} T_2'$ for some T_1', T_2'. By Lemma 53, we have $E_1 \xrightarrow{a} E_1'\mathcal{R} \sim T_1'$ and $E_2 \xrightarrow{\overline{a}} E_2'\mathcal{R} \sim T_2'$ for some E_1', E_2'. Thus

  $$E_1 \,|\, E_2 \,|\, \cdots \,|\, E_k \xrightarrow{\tau} E_1' \,|\, E_2' \,|\, \cdots \,|\, E_k\mathcal{R} \sim T_1' \,|\, T_2' \,|\, \cdots \,|\, T_k \sim T'.$$

  For the third case, it suffices to notice that

  $$E_1 \,|\, E_2 \,|\, \cdots \,|\, E_k \;\mathcal{R}\; T_1' \,|\, T_2 \,|\, \cdots \,|\, T_k.$$

This completes the proof. □

References

[1] S. Abramsky. The Lazy Lambda Calculus. D. Turner (eds.): *Declarative Programming*, 65-116, Addison-Wesley, 1988.

[2] M. Alsuwaiyel. *Algorithm Design Techniques and Analysis*. World Scientific, 1999.

[3] R. Amadio, I. Castellani and D. Sangiorgi. On Bisimulations for the Asynchronous π-calculus. U. Montanari, V. Sassone (eds.): *CONCUR 1996*, Lecture Notes in Computer Science **1119**: 147-162, 1996.

[4] J. Baeten and W. Weijland. *Process Algebra*, Cambridge Tracts in Theoretical Computer Science **18**, CUP, 1990.

[5] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, Studies in Logic and Foundations of Mathematics, North-Holland, 1984.

[6] G. Boudol. Asynchrony and the π-calculus. Technical Report RR-1702, INRIA Sophia-Antipolis, 1992.

[7] N. Busi, M. Gabbrielli and G. Zavattaro. Replication vs Recursive Definitions in Channel Based Calculi. *ICALP 2003*, Lecture Notes in Computer Science **2719**: 133-144, 2003.

[8] N. Busi, M. Gabbrielli and G. Zavattaro. Comparing Recursion, Replication and Iteration in Process Calculi. *ICALP 2004*, Lecture Notes in Computer Science **3142**: 307-319, 2004.

[9] N. Busi and G. Zavattaro. On the Expressive Power of Movement and Restriction in Pure Mobile Ambients. *Theoretical Computer Science*, **322**: 477-515, 2004.

[10] D. Cacciagrano, F. Corradini, J. Aranda and F. Valencia. Linearity, Persistence and Testing Semantics in the Asynchronous Pi-Calculus. *ENTCS* **194**: 59-84, 2008.

[11] X. Cai and Y. Fu. The λ-Calculus in the π-Calculus, 2009. The paper is downloadable at `http://basics.sjtu.edu.cn/~ yuxi/papers`.

[12] L. Cardelli and A. Gordon. Mobile Ambients. *Theoretical Computer Science*, **240**: 177-213, 2000.

[13] B. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[14] R. De Nicola, M. Hennessy. Testing Equivalence for Processes. *Theoretical Computer Science*, **34**: 83-133, 1984.

[15] X. Dong and Y. Fu. Extensional Net Theory. 2009.

[16] P. van Emde Boas. Machine Models and Simulaitons. In *Handbook of Theoretical Computer Science: Algorithm and Complexity*, volume A, ed. by J. van Leeuwen, Elservier, 1990.

[17] A. Finkel and Ph. Schnoebelen. Well-Structured Transition System Everywhere. *Theoretical Computer Science*, **256**: 63-92, 2001.

[18] Y. Fu. On Quasi Open Bisimulation. *Theoretical Computer Science*, **338**: 96-126, 2005.

[19] Y. Fu. Fair Ambients. *Acta Informatica*, **43**: 535-594, 2007.

[20] Y. Fu. Theory of Interaction. Working paper, 2009.

[21] Y. Fu. Theory of Value Passing Calculus. Working paper, 2009.

[22] Y. Fu and Z. Yang, Tau Laws for Pi Calculus, *Theoretical Computer Science*, **308**: 55-130, 2003.

[23] Y. Fu, H. Zhu. Theory of Name Passing Calculus. Working paper, 2009.

[24] J. Girard, Linear Logic, *Theoretical Computer Science*, **50**: 1-102, 1987.

[25] R. van Galbbeek. Linear Time – Branching Time Spectrum II. *CONCUR'93*, Lecture Notes in Computer Science **715**: 66-81, 1993.

[26] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Information Processing'89*, North-Holland, 613-618, 1989.

[27] P. Giambiagi, G. Schneider and F. Valencia. On the Expressiveness of Infinite Behavior and Name Scoping in Process Calculi. *FOSSACS 2004*, Lecture Notes in Computer Science **2987**: 226-240, 2004.

[28] D. Gorla. Towards a Unified Approach to Encodability and Separation Results for Process Calculi. *CONCUR 2008*, Lecture Notes in Computer Science 5201, 492-507, 2008.

[29] D. Gorla. Comparing Communication Primitives via their Relative Expressive Power, *Information and Computation* **206**: 931-952, 2008.

[30] D. Gorla. On the Relative Power of Ambient-based Calculi. *TGC'08*, Lecture Notes in Computer Science **5474**: 141-156, 2009.

[31] D. Gorla. On the Relative Power of Calculi for Mobility. *25th International Conference on Mathematical Foundation of Programming Semantics*, ENTCS, Elsevier, 2009.

[32] M. Hennessy. An Algebraic Theory of Processes. MIT Press, Cambridge, MA, 1988.

[33] M. Hennessy, H. Lin, Symbolic Bisimulations. *Theoretical Computer Science*, **138**: 137-161, 1995.

[34] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communications. M. Tokoro, O. Nierstrasz, P. Wegner and A. Yonezawa (eds.), *ECOOP 1991*, Geneva, Switzerland, Lecture Notes in Computer Science **512**: 133-147, 1991.

[35] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Company, 1979.

[36] A. Ingolfsdottir and H. Lin. A Symbolic Approach to Value-passing Processes, Chapter 7 of *Handbook of Processes Algebra*, Elsevier, 2001.

[37] J. Kruskal. The Theory of Well Ordering: A Frequently Discovered Concept, *Journal of Combinatorial Theory, Series A*, **13**: 297-305, 1972.

[38] I. Lanese, J. Perez, D. Sangiorgi and A. Schmitt. On the Expressiveness and Decidability of Higher-Order Process Calculi. *LICS'08*, 145-155, 2008.

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

[40] R. Milner. Functions as Processes. *Mathematical Structures in Computer Science*, **2**: 119-146, 1992.

[41] R. Milner. Elements of Interaction. *Communication of the ACM*: 78-89,1993.

[42] S. Maffeis and I. Phillips. On the Computational Strength of Pure Ambient Calculi. *Theoretical Computer Science*, **330**: 501-551, 2005.

[43] R. Milner, J. Parrow and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, **100**: 1-40 (Part I), 41-77 (Part II), Academic Press, 1992.

[44] R. Milner, The Polyadic π-Calculus: a Tutorial, *Proceedings of the 1991 Marktoberdorf Summer School on Logic and Algebra of Specification*, NATO ASI, Series F, Springer-Verlag, 1993.

[45] R. Milner and D. Sangiorgi, Barbed Bisimulation, *ICALP 1992*, Lecture Notes in Computer Science **623**, 685-695, 1992.

[46] U. Nestmann and B. Pierce. Decoding Choice Encodings. In: U. Montanari, V. Sassone (eds.): *CONCUR 1996*, Lecture Notes in Computer Science **1119**: 179-194, 1996.

[47] U. Nestmann. What is a Good Encoding of Guarded Choices? *Information and computation*, **156**: 287-319, 2000.

[48] U. Nestmann. Welcome to the Jungle: A Subjective Guide to Mobile Process Calculi, In: C. Baier and H. Hermanns (eds.): *CONCUR 2006*, Lecture Notes in Computer Science **4137**: 52-63, 2006.

[49] C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculus. *Mathematical Structures in Computer Science*, **13**: 685-719, 2003.

[50] D. Park. Concurrency and Automata on Infinite Sequences. *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, Lecture Notes in Computer Science **104**: 167-183, Springer, 1981.

[51] I. Phillips and M. Vigliotti. Leader Election in Rings of Ambient Processes, *Theoretical Computer Science*, **356**: 468-494, 2006.

[52] L. Priese. On the Concept of Simulation in Asynchronous, Concurrent Systems. *Progress in Cybernatics and Systems Research*, vol.VII, 85-92, Hemisohere, Dubl. Corp., 1978.

[53] C. Palamidessi, V. Saraswat, F. Valencia and B. Victor. On the Expressiveness of Linearity vs Persistence in the Asynchronous Pi Calculus. *LICS 2006*, 59-68, 2006.

[54] D. Sangiorgi. π-calculus, Internal Mobility and Agent-Passing Calculi. *Theoretical Computer Science*, **167**: 235-274, 1996.

[55] D. Sangiorgi, A Theory of Bisimulation for π-Calculus, *Acta Informatica*, **3**: 69-97, 1996.

[56] D. Sangiorgi. Bisimulation for Higher Order Process Calculi. *Information and Comptation*, **131**: 141-178, 1996.

[57] D. Sangiorgi. On the Origin of Bisimulation and Coinduction. *ACM Transactions on Programming Languages and Systems*, **31**, 2009.

[58] M. Sipser. Introduction to Theory of Computation. PWS Publishing Company, 1997.

[59] D. Sangiorgi and D. Walker. On Barbed Equivalence in π-Calculus, In: K. Larsen and M. Nielsen (eds.): *CONCUR 2001*, Lecture Notes in Computer Science **2154**: 292-304, 2001.

[60] D. Sangiorgi and D. Walker. *The Pi Calculus–A Theory of Mobile Processes*, CUP, 2001.

[61] B. Thomsen. A Theory of Higher Order Communicating Systems. *Information and Computation*, **116**: 38-57, 1995.

[62] M. Vigliotti, I. Phillips and C. Palamidessi. Tutorial on Separation Results in Process Calculi via Leader Election. *Theoretical Computer Science*, **388**: 267-289, 2007.

[63] D. Walker. Bisimulation and Divergence. *Information and Computation*, **85**: 202-241, 1990.

[64] I. Wegener. *Complexity Theory*, Spronger-Verlag, 2005.