The Responsive Bisimulations in the κ-calculus

UNSW-CSE-TR-0205

Xiaogang Zhang and John Potter
School of Computer Science and Engineering
University of New South Wales, Australia
{xzhang,potter}@cse.unsw.edu.au

Abstract

Ongoing work attempts to model concurrent object systems using process algebra. The behaviour of an object can be described as the composition of a process representing the basic functionality of the object and separate processes controlling the concurrent behaviour of that object. While familiar usually failed, the responsive bisimulation proposed by the authors in an earlier paper where the delaying a message locally and remotely have the same effect as long as potential interference by competing receptors is avoided, is able to capture the behavioural equivalence between object components. With this bisimulation, an equivalence between the π-calculus expression $(\nu n)(m.\overline{n} \mid k.n.P)$ *and* $k.m.P$ *then can be achieved. However, in the earlier paper, the responsive bisimulation was described in the polar π-calculus, which added a few improved features for modelling concurrent objects while maintains the syntatical simplicity similar to the normal π-calculus, but is still difficult to express general behaviurs of concurrent objects efficiently. The κ-calculus, where locks are included as primitive, in the other hand, is more expressive and flexible in modelling compositional concurrenct objects.*

This paper presents responsive bisimulation in the κ-calculus, and therefore will form an improved base for studies on both the theory of behaviours composition and the semantics of compositional concurrent OO programming languages.

The Responsive Bisimulations in the κ-calculus

Xiaogang Zhang and John Potter
School of Computer Science and Engineering
University of New South Wales, Australia
{xzhang,potter}@cse.unsw.edu.au

Abstract

Ongoing work attempts to model concurrent object systems using process algebra. The behaviour of an object can be described as the composition of a process representing the basic functionality of the object and separate processes controlling the concurrent behaviour of that object. While familiar usually failed, the responsive bisimulation proposed by the authors in an earlier paper where the delaying a message locally and remotely have the same effect as long as potential interference by competing receptors is avoided, is able to capture the behavioural equivalence between object components. With this bisimulation, an equivalence between the π-calculus expression $(\nu n)(m.\overline{n} \mid k.n.P)$ *and* $k.m.P$ *then can be achieved. However, in the earlier paper, the responsive bisimulation was described in the polar π-calculus, which added a few improved features for modelling concurrent objects while maintains the syntatical simplicity similar to the normal π-calculus, but is still difficult to express general behaviurs of concurrent objects efficiently. The κ-calculus, where locks are included as primitive, in the other hand, is more expressive and flexible in modelling compositional concurrenct objects.*

This paper presents responsive bisimulation in the κ-calculus, and therefore will form an improved base for studies on both the theory of behaviours composition and the semantics of compositional concurrent OO programming languages.

1 Introduction

With the ability to directly model dynamic reference structures, process algebra such as the π-calculus ([Milner92], [Milner96]) and its variations have been applied to modelling concurrent object systems ([Walker95], [Jones93], [Sangiorgi96], [Hüttel96], [Zhang97]). Some researchers ([Schneider97], [Zhang98A], [Zhang98B]) have also applied it in modelling compositional objects in aspect-oriented programming style ([Aksit92], [Holmes97]) to avoid the inheritance anomaly [McHale94].

With the idea of [Zhang98A] and [Zhang98B] in modelling concurrent objects in the π-calculus, the behaviour of a concurrent object can be modelled as the parallel composition of two processes: a process F which represents the object's functional behaviour and can be expressed with the generic form $F \stackrel{\text{def}}{=} \prod !n_i(\tilde{x}).M_i\langle\tilde{x}\rangle$, and a process C which represents the constraints on the object's concurrent behaviour. In effect, F on its own, represents an object with no constraints on its concurrent interactions. For example, the functionality of a buffer object can be described by the expression $F_B \stackrel{\text{def}}{=} !n_r(x).M_r\langle x\rangle \mid !n_w(x).M_w\langle x\rangle$, where $n_r(x).M_r\langle x\rangle$ and $n_w(x).M_w\langle x\rangle$ represent the behaviour of the read and write methods respectively, each of them can have unlimited invocations executing in parallel without any concern of interfering among them. To discipline those invocations, assume a synchronisation behaviour modelled by the control process $C_s \stackrel{\text{def}}{=} m_r(x).\overline{n}_r\langle x\rangle + m_w(x).\overline{n}_w\langle x\rangle$, where the sum operator in fact represents a mutual exclusion lock on those methods. Then the parallel composition of the two processes, $(\nu n)(C_s \mid F)$, will be weakly bisimilar to $R_s \stackrel{\text{def}}{=} m_r(x).M_r\langle x\rangle + m_w(x).M_w\langle x\rangle$, as expected. However, there are two problems need to be solved.

The first problem is, the equivalence between the expected behaviour and the composed behaviour cannot be always captured by familiar bisimulation relations. For example, the equality between processes $(\nu n)(m.\overline{n} \mid n.P)$ and $m.P$ is not recognised by most known bisimulations. The necessary of this kind equivalence can be shown by the following "real world" communication example:

> In the mailroom of a business skyscraper, the property manager uses internal mail to send bills to her tenants and collect payments. Each tenant has a locked mailbox, which located either on the mailroom wall and can be opened from outside of the mailroom by the tenant, or on the door of the tenant's suite and a postman delivers mails from mailroom to the tenant's suite. For the property manager, whether a tenant is classified as behaving "good" or "bad" should only depend on whether he pays the bill on time and in cash, and where the tenant's mailbox locates should make no difference. The manager needs only to monitor the arrival of payments to identify the tenants' behaviour.

To describe a little bit more formally, let the process O_1 and O_2 illustrated in Figure 1-1 represent two different versions of the internal structure of the same composed object in a state where its only method is blocked by the lock of key κ (e.g., the key for a mailbox). The only difference between them is that O_1 has an extra "empty" control $Ctrl_e$ (postman) which does nothing but forwards whatever message received from channel m to the next control $Ctrl_l$ (locked mailbox). The body (tenant) of these two can always give the same response (payment) if fed with the same message (bill). If an unlocking signal is received via channel κ, both O_1 and O_2 can accept incoming messages and process them immediately. If some message arrives before the unlocking, O_1 will store it in an internal buffer (door mailbox) and delay the process until unlocked, but O_2 will leave the message in the external buffer (mailroom) as it was, while waiting for unlocking.

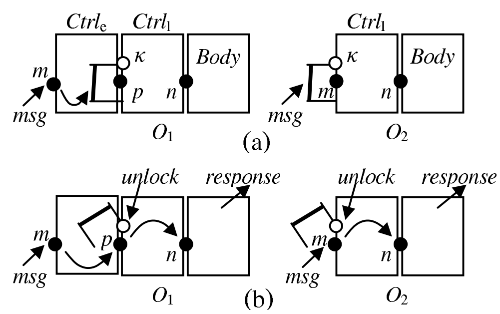


Figure 1-1

For a client (property manager) who is sending the message, the behaviour of the target object can be measured only by observing how it responds. Therefore, the behaviour of O_1 and O_2 are identical in the client's eyes, since the responses they can give are the same (both from the same *Body*). However, this behavioural similarity cannot be captured by most of the known behavioural equivalence relations, since in some stage O_1 can perform an input action from the channel m while O_2 cannot. Even the weak barbed-equivalence, one of the weakest, is too strong for them, since $O_1 | R$ and $O_2 | R$ are not weakly barbed-bisimilar for some R, such as $R \stackrel{\text{def}}{=} \overline{m}\langle a \rangle$.

To solve this problem, [Zhang01A] proposed the notion of responsive bisimulation, where only "localise" testing message is considered while measures the behaviour of the target process by observing its response. A "localise" message permits only the target process to access, even when the communication channel is visible globally. Therefore the responsive bisimulation filters out the environmental effects, and can capture the similarity of responsive behaviours of object processes, and more interestingly, the general behaviour of control processes. Furthermore, these relations allow the behavioural composition of objects to be studied more easily, since we are able to derive the equivalence of a larger collection of behaviours. For example, let $C \succ F$ stands for the operation which composes an object component process F and a control process C, a special kind of object component process, to yield a new object component process with expected behaviour. With the responsive bisimulation relation, we not only have the associative law, i.e. $C_1 \succ (C_2 \succ F) \equiv (C_1 \succ C_2) \succ F$, but also the identity law, ie., there is some empty control (identity) E such that for all F satisfying $E \succ F$, the composed object $E \succ F$ is equivalent to the original object F, and for all control process C satisfying either $E \succ C$ or $C \succ E$, the three control processes $E \succ C$, $C \succ E$ and C are all equivalent ([Zhang01C]).

In [Zhang01A] the responsive bisimulation was studied using the *polar π-calculus* as the mathematical tool, which adoptes the concept of polarised names from [Odersky95a], and then syntatically added the restriction that only the output polar of a name may be transmitted by communication. Both these features match the nature of object-oriented systems.

The second problem is that, the general exclution relations between object methods are difficult to be presented efficiently and compositionally in the π-calculu and most variations, including the polar π-calculus used by [Zhang01A]. For example, assume an object with three methods m_1, m_2 and m_3, and assume that from the exclusive requirements, mutually exclusive should be maintained between method m_1 and m_2, and fully concurrent execution is allowed between method m_1 and m_3. These two requirements may be modelled in the π-calculu respectively as the two control processes $C_1 \stackrel{\text{def}}{=} m_1(\tilde{x}).\overline{n_1}\langle\tilde{x}\rangle + m_2(\tilde{x}).\overline{n_2}\langle\tilde{x}\rangle$ and $C_2 \stackrel{\text{def}}{=} m_1(\tilde{x}).\overline{n_1}\langle\tilde{x}\rangle \mid !m_3(\tilde{x}).\overline{n_3}\langle\tilde{x}\rangle$.

Now consider the following different cases on addition requirements

1) Fully concurrent execution is also allowed between method m_2 and m_3, i.e., $C_3 \stackrel{\text{def}}{=} m_2(\tilde{x}).\overline{n_2}\langle\tilde{x}\rangle \mid\, !m_3(\tilde{x}).\overline{n_3}\langle\tilde{x}\rangle$;
2) Mutually exclusive should be maintained between method m_2 and m_3, i.e., $C_3 \stackrel{\text{def}}{=} m_2(\tilde{x}).\overline{n_2}\langle\tilde{x}\rangle + m_3(\tilde{x}).\overline{n_3}\langle\tilde{x}\rangle$;
3) The same as 1), except that m_1 is a reading method, and should not mutually exclusive with itself, but must be mutually exclusive with the writing method m_2.

For 1), it is easy to put the addition requirement together with the previous ones to construct a composed control process: $C_s \stackrel{\text{def}}{=} (m_1(\tilde{x}).\overline{n_1}\langle\tilde{x}\rangle + m_2(\tilde{x}).\overline{n_2}\langle\tilde{x}\rangle) \mid\, !m_3(\tilde{x}).\overline{n_3}\langle\tilde{x}\rangle$. However, it is difficult for 2) and 3), because:

a) The entire control C_s has to be rewritten from scratch, without re-using of the previous controls;
b) The expression of new control becomes extremely complicated and crummy, difficult to read or even write;
c) The expression of exclusion constraint may not be able to be written in a generic and unified or abstract form.

In contrary, the algebra of exclusion proposed by [Noble00] can express all those easily and efficiently, for example, the above three situations can be described in turn as $\overline{m}_1\times\overline{m}_2 \mid m_3$, $\overline{m}_1\times\overline{m}_2 \mid \overline{m}_2\times\overline{m}_3$, and $m_1\times\overline{m}_2 \mid \overline{m}_3$.

However, the algebra of exclusion is static express, unable to present the dynamical behaviour of concurrent objects. To solve this problem, [Zhang01B] proposed an extended calculus, the κ-calculus which welds the mobility power of the π-calculus with the synchronisation expressiveness of the algebra of exclusion ([Noble00]). With the κ-calculus, a control C_s will have the form $C_s \stackrel{\text{def}}{=} (\tilde{m},\tilde{n})\lfloor\,\rfloor\circ[\mathbb{G}\langle\!\langle\{S_i\langle\overline{n}_i\rangle_{i\in I}\}\rangle\!\rangle\langle\tilde{m}\rangle]$, where $\mathbb{G}$ specifies the exclusion relations, and each S_i gives some scheduling information such as unlocking or early return on the ith method. As the example, for each of the previous mentioned three situations we may write the $\mathbb{G}$ as:

1) $!(\nu\,\kappa)\dagger m_1(\tilde{x})\check{\kappa}@\{\dagger m_1,\dagger m_2\}.\,\eta_1\langle\tilde{x},\hat{\kappa}\rangle \;\otimes\; !(\nu\,\kappa)\dagger m_2(\tilde{x})\check{\kappa}@\{\dagger m_1,\dagger m_2\}.\,\eta_2\langle\tilde{x},\hat{\kappa}\rangle \;\otimes\; !(\nu\,\kappa)\dagger m_3(\tilde{x})\check{\kappa}@\emptyset.\,\eta_3\langle\tilde{x},\hat{\kappa}\rangle$;
2) $!(\nu\,\kappa)\dagger m_1(\tilde{x})\check{\kappa}@\{\dagger m_1,\dagger m_2\}.\,\eta_1\langle\tilde{x},\hat{\kappa}\rangle \;\otimes\; !(\nu\,\kappa)\dagger m_2(\tilde{x})\check{\kappa}@\{\dagger m_1,\dagger m_2,\dagger m_3\}.\,\eta_2\langle\tilde{x},\hat{\kappa}\rangle \;\otimes\; !(\nu\,\kappa)\dagger m_3(\tilde{x})\check{\kappa}@\{\dagger m_2,\dagger m_3\}.\,\eta_3\langle\tilde{x},\hat{\kappa}\rangle$;
3) $!(\nu\,\kappa)\dagger m_1(\tilde{x})\check{\kappa}@\{\dagger m_2\}.\,\eta_1\langle\tilde{x},\hat{\kappa}\rangle \;\otimes\; !(\nu\,\kappa)\dagger m_2(\tilde{x})\check{\kappa}@\{\dagger m_1,\dagger m_2\}.\,\eta_2\langle\tilde{x},\hat{\kappa}\rangle \;\otimes\; !(\nu\,\kappa)\dagger m_3(\tilde{x})\check{\kappa}@\emptyset.\,\eta_3\langle\tilde{x},\hat{\kappa}\rangle$;

As we can see here, the κ-calculus can not only solve all the problems we have pointed, but also provider extra ability in behaviour separation, -- the separation of S_i from $\mathbb{G}$.

In this paper we put the two pieces together, present the responsive bisimulation in the κ-calculus, and therefore form a full base for studing of compositional concurrrent objects and the theory of composition.

The rest of the paper is structured as follows: section 2 briefly introduces the κ-calculus and related notions, section 3 defines responsive bisimulation, section 4 gives some properties of the equivalences and other theoretical results, section 5 discusses some further issues relating the responsive bisimulation with other notions, section 6 briefly describes some applications of responsive bisimulation in modelling compositional objects with related results, and section 7 concludes the paper.

2 The κ-calculus

The κ-calculus ([Zhang01B]) is a process calculus especially suitable for modelling the composition behaviours of concurrent objects. Like the asynchronous π-calculus ([Amadio96]), it uses asynchronous communication, i.e. an output action does not block other actions. Like the polar π-calculus ([Zhang01A]), it adopts the concept of polarised names ([Odersky95a]), and the restriction that only output polar of a name can be transmitted by communication. In addition, close to [Liu97], [Philippou96], [Zhang98A] and [Zhang98B], the κ-calculus has a higher-order extension which is only involved with higher-order process abstractions but excludes higher-order communication ([Sangiorgi92a], [Sangiorgi92b]), and therefore can employ the relatively simpler bisimilarity theory of the π–calculus while providing more power on behaviour separation.

The major significance in the κ-calculus is the inclusion of lock as primitive. In the conventional CCS or π-calculi, input-guarded processes can only be composed to play either a "one be chosen then all others have to die" game in the mutually exclusive choice (the sum operation "+"), or "no one minds others' business" game in the parallel composition "|". In the guarded exclusive choice of the κ-calculus, however, the exclusion between branches are explicitly defined, and the invocation of an input action can cause a lock on pre-specified branches, which may become available again when the lock is released. The “+” and “|” operations then are unified into the guarded exclusive choice as two extreme cases. This enables the κ-calculus to obtain the expressibility of the algebra of exclusion ([Noble00]) for methods exclusion of concurrent objects, allows the separation of some major concurrency behaviours of objects to be presented in a much more natural and clearer way. The κ-calculus distinguishes the labels for communication channel names and that for locking keys, in order to provent cross using by mistake.

2.1 The syntax of the κ-calculus

In the κ-calculus we distinguish two disjoint sets of label names, the communication channel names, and the key names for locking. Let $\mathcal{M}$ be the set of all communication channel names, ranged over by expressions m,u,v and variables x,y. Let ${}^{+}\mathcal{M} \stackrel{def}{=} \{{}^{+}m\colon m\in\mathcal{M}\}$ and ${}^{-}\mathcal{M} \stackrel{def}{=} \{{}^{-}m\colon m\in\mathcal{M}\}$ be the sets of input polar and output polar of all channel names respectively. Let $\mathcal{K}$ be set of all release keys of locking, ranged over by κ. Let ${}^{+}\mathcal{K} \stackrel{def}{=} \{\check{\kappa}\colon \kappa\in\mathcal{K}\}$ and ${}^{-}\mathcal{K} \stackrel{def}{=} \{\hat{\kappa}\colon \kappa\in\mathcal{K}\}$ be the sets of input polar and output polar of all keys respectively. Then the set of all label names is $\mathcal{N} \stackrel{def}{=} \mathcal{M}\cup\mathcal{K}$, ranged over by n. Consequently, we have various sets of polars, such as ${}^{\pm}\mathcal{M} \stackrel{def}{=} {}^{+}\mathcal{M}\cup{}^{-}\mathcal{M}$, ${}^{\pm}\mathcal{K} \stackrel{def}{=} {}^{+}\mathcal{K}\cup{}^{-}\mathcal{K}$, ${}^{+}\mathcal{N} \stackrel{def}{=} {}^{+}\mathcal{M}\cup{}^{+}\mathcal{K}$, ${}^{-}\mathcal{N} \stackrel{def}{=} {}^{-}\mathcal{M}\cup{}^{-}\mathcal{K}$, and ${}^{\pm}\mathcal{N} \stackrel{def}{=} {}^{+}\mathcal{N}\cup{}^{-}\mathcal{N}$. Let $a,b\in{}^{\pm}\mathcal{N}$ be polar constants, and $w\in{}^{\pm}\mathcal{N}$ be polar variables. Let both $\tilde{r}$ and $\{r_{i\in I}\}$, where I is an index set of arity n, be abbreviations for $r_1,r_2,\ldots,r_n$. The generic process terms P in the κ-calculus are generated by the following grammars:

$$P ::= \mathbf{0_P} \mid {}^{-}m\langle\tilde{u}\rangle \mid \hat{\kappa} \mid (\nu\,\tilde{n})P \mid P_1 \mid P_2 \mid \Lambda\circ[G] \mid A\langle\tilde{a}\rangle \mid \eta, \qquad G ::= B \mid (\nu\,\tilde{n})G \mid G_1\otimes G_2 \mid D\langle\tilde{a}\rangle \mid \mathscr{G}«\tilde{P}», \qquad B ::= \mathbf{0_G} \mid !\beta.P \mid !(\nu\,\kappa)\beta.P$$

$$A ::= (\tilde{w})P, \qquad D ::= (\tilde{w})G, \qquad \mathscr{G} ::= «\tilde{\eta}»G, \qquad \beta ::= {}^{+}m(\tilde{x})L, \qquad L ::= \check{\kappa}@J \mid (\nu)@J, \qquad J ::= \{{}^{+}\tilde{m}\} \mid \varnothing \mid \boldsymbol{M}$$

The set of all actions a process may take can be specified by $\alpha ::= {}^{+}m(\tilde{u}) \mid (\nu\,\tilde{v}){}^{-}m\langle\tilde{u}\rangle \mid \hat{\kappa} \mid \check{\kappa} \mid \tau$, where $\tilde{v}\subseteq\tilde{u}$ and $m\notin\tilde{v}$.

Most process terms (P–terms) are similar to those in normal π-calculi: $\mathbf{0_P}$ is the inactive (terminated) process; ${}^{-}m\langle\tilde{u}\rangle$ is the output action which sends output polars $\tilde{u}$ into the channel m; $(\nu\,\tilde{n})P$ binds the set of labels $\tilde{n}$, and therefore both polars of each of them, within the scope of P; $P_1 \mid P_2$ indicates two processes run in parallel; $A\langle\tilde{a}\rangle$ is an instance of parameterised process agent, giving the process agent abstraction $A \stackrel{def}{=} (\tilde{w})P$ is obeying $((\tilde{w})P)\langle\tilde{a}\rangle \equiv P\{\tilde{a}/\tilde{w}\}$; η is a process variable. For the rest two terms, $\hat{\kappa}$ is the action which emits the *unlock signal* within the scope where the name κ is bound; and $\Lambda\circ[G]$ is the *guarded exclusive choice* (*GEC* choice), where G defines the exclusion behaviour which can place some locks on the process itself, and Λ records the lock status. For the choice terms (G–terms), B is a choice branch; $G_1\otimes G_2$ is the choice composition; $(\tilde{w})G$ and $D\langle\tilde{a}\rangle$ are abstraction and instance of choice agent respectively, obeying $((\tilde{w})G)\langle\tilde{a}\rangle \equiv G\{\tilde{a}/\tilde{w}\}$; higher order G-term agent $\mathscr{G} \stackrel{def}{=} «\tilde{\eta}»G$ accepts processes as parameters in the double angled brackets, and obeying $(«\tilde{\eta}»G)\,«\tilde{P}» \equiv G\{\tilde{P}/\tilde{\eta}\}$; $\mathbf{0_G}$ is the unreachable choice, in the future we can omit the subscript of both $\mathbf{0_G}$ and $\mathbf{0_P}$ without any ambiguity. Unlike that in π, every branch B here always behaves as a (lazy) replication, among them, “$!(\nu\,\kappa)$” creates a fresh key κ private to each replicated copy; in $\beta.P$ the action prefix operator “.” indicates the execution of action β before the execution of the continuation process P; the action ${}^{+}m(\tilde{x})L$, where we stipulate that $\{\tilde{x}\}\cap n(L) = \varnothing$, produces two simultaneous events: receiving information $\tilde{x}$ from the input port of channel m, and triggering the lock L; the lock $L=\check{\kappa}@J$ read as “lock all input channels in J with key κ”, where the exclusion set J specifies the channels to be locked within the *GEC* choice and κ is the key for unlocking the lock; abbreviation $L=(\nu)@J$ indicates a lock with an anonymous key, that is, $!{}^{+}m(\tilde{x})(\nu)@J.P \equiv !(\nu\,\kappa){}^{+}m(\tilde{x})\check{\kappa}@J.P$ for $\kappa\notin fn(P)$, in other words, it is an unreleasable lock; $\boldsymbol{M}$ is the entire ${}^{+}\mathcal{M}$, the set of input polar of all channel names, and therefore enforces the locking of every channel within the *GEC* choice. The other part of the *GEC* choice, Λ, acts as a state machine maintaining and monitoring the current status of locks, and is described in an independent language. Different Λ grammars will give different locking schemes and locking status evolution paths, but will not interfere with semantic or syntax of the G language, and vice versa. In one of the simplest such locking scheme, where duplicate locks upon the same channel with the same key will have the same effect as such a single lock ([Zhang01B]), is defined by the grammar $\Lambda ::= \lfloor\,\rfloor \mid \lfloor\tilde{L}\rfloor \mid \Lambda\Lambda$ and the structural equivalencies rules are shown in Figure 2-1.

lstr-SMM (Summation) : $\lfloor\,\rfloor\Lambda \equiv \Lambda;\quad \Lambda_1\Lambda_2 \equiv \Lambda_2\Lambda_1;\quad \Lambda_1(\Lambda\Lambda_2) \equiv (\Lambda_1\Lambda)\Lambda_2.$

lstr-EMP (Empty lock) : $\lfloor\check{\kappa}@\varnothing\rfloor \equiv \lfloor\,\rfloor;$

lstr-LKC (Combination) : $\lfloor\check{\kappa}@J_1, \check{\kappa}@J_2\rfloor \equiv \lfloor\check{\kappa}@(J_1\cup J_2)\rfloor;$

lstr-GRP (Grouping) : $\lfloor\tilde{L}_1\rfloor\lfloor\tilde{L}_2\rfloor \equiv \lfloor\tilde{L}_1,\tilde{L}_2\rfloor;$

Figure 2-1 Structual equivalence of locking status terms

Notation 2−1: Some auxiliary operations/functions (the formal definitions can be found in [Zhang01B]) are needed for integrating a Λ language into the κ-calculus:

- $guard(G)$: gives the set of all branches' input prefix channel names in G, and defined by
  $guard(!\dagger m(\tilde{x})L.P) \stackrel{\text{def}}{=} m$; $\quad guard((\nu\,\tilde{n})G) \stackrel{\text{def}}{=} guard(G)$; $\quad guard(G_1 \otimes G_2) \stackrel{\text{def}}{=} \{guard(G_1)\} \cup \{guard(G_2)\}$.;
- $lock(J,\kappa,\Lambda)$: gives the truth value for whether Λ indicates all the input polars appeared in J are locked by κ;
- $lset(\Lambda)$: gives the set of all channel names for which their input polars are indicated by Λ as locked;
- $keys(\Lambda)$: gives the set of all key κ for which there exists some $J \neq \varnothing$ such that $lock(J,\kappa,\Lambda)$=true;.
- $addl(L,\Lambda)$: gives the new locking status after adding L to the original locking status Λ.
- Λ/L : gives the new locking status after removing L from the original locking status Λ.

We usually use $\Lambda \stackrel{\text{def}}{=} \lfloor\,\rfloor$ to represent an empty lock, and for any Λ language, we always require that:

$lock(J,\kappa,\lfloor\,\rfloor) \equiv False$, $\quad lset(\lfloor\,\rfloor) \equiv \varnothing$, $\quad keys(\lfloor\,\rfloor) \equiv \varnothing$, $\quad addl(L,\lfloor\,\rfloor) \equiv \lfloor L \rfloor$ and $\lfloor\,\rfloor/L \equiv \lfloor\,\rfloor$.

Notation 2−2: If $m \notin lset(\Lambda)$, we say that Λ *allows the commitment* on $\dagger m$, denoted as $\Lambda \downarrow \dagger m$;
if $m \in lset(\Lambda)$, we say that Λ *blocks* the channel $\dagger m$, denoted as $\Lambda \not\downarrow \dagger m$.
If for some $J' \subseteq J$, $J' \neq \varnothing$ and $lock(J', \kappa, \Lambda)$, we say that Λ can commit $\check{\kappa}@J$, denoted as $\Lambda \downarrow \check{\kappa}@J$;
otherwise we say that he Λ cannot commit on $\check{\kappa}$ over J, denoted as $\Lambda \not\downarrow \check{\kappa}@J$.
If for some $J \supseteq lset(\Lambda)$, $\Lambda \downarrow \check{\kappa}@J$, we say that Λ can commit $\check{\kappa}$, denoted as $\Lambda \downarrow \check{\kappa}$;
otherwise we say that he Λ cannot commit on $\check{\kappa}$, denoted as $\Lambda \not\downarrow \check{\kappa}$.
In the form of labelled transition, we denote $\Lambda \xrightarrow{\dagger m \uparrow L} \Lambda'$ for $\dagger m \notin lset(\Lambda)$ and $\Lambda' = addl(L,\Lambda)$; and
$\Lambda \xrightarrow{\check{\kappa}@J} \Lambda'$ for $\Lambda \downarrow \check{\kappa}@J$ and $\Lambda' = \Lambda/\kappa@J$.

Notation 2−3: Similar to the polar π-calculus, besides the functions *fn*, *bn* and *n* for identifying the sets of free, bound and all names respectively of a P–term, G–term or action, we also use more specified functions, such as *fin*, *bin*, *in*, *fon*, *bon* and *on* to identify free, bound and all input or out polars. Further more, as in the κ-calculus we distinguish communication channel names and keys, we also use finer grained functions, *fnc*, *bnc*, *nc*, *finc*, *binc*, *inc*, *fonc*, *bonc* and *onc* for communication channels only, and *fnk*, *bnk*, *nk*, *fink*, *bink*, *ink*, *fonk*, *bonk* and *onk* for keys only.

Notation 2−4: The following process abbreviations are for convenience and can simplify expressions:

$$\dagger m(\tilde{u}).P \stackrel{\text{def}}{=} \lfloor\,\rfloor \circ [\, !\dagger m(\tilde{u})(\nu)@[\dagger m].P \,], \quad \prod \dagger m_i(\tilde{u}).P_i \equiv \lfloor\,\rfloor \circ [\, \bigotimes !\dagger m_i(\tilde{u})(\nu)@[\dagger m_i].P_i \,], \quad \sum \dagger m_i(\tilde{u}).P_i \stackrel{\text{def}}{=} \lfloor\,\rfloor \circ [\, \bigotimes !\dagger m_i(\tilde{u})(\nu)@\boldsymbol{M}.P_i \,]$$

$$!\dagger m(\tilde{u}).P \stackrel{\text{def}}{=} \lfloor\,\rfloor \circ [\, !\dagger m(\tilde{u})(\nu)@\varnothing.P \,], \qquad \prod !\dagger m_i(\tilde{u}).P_i \equiv \lfloor\,\rfloor \circ [\, \bigotimes !\dagger m_i(\tilde{u})(\nu)@\varnothing.P_i).$$

These abbreviations give an illustration of that for input-prefixed processes, the standard parallel and sum compositions in conventional π-calculus become special cases of GEC choice in the κ-calculus. Further more, as these abbreviations suggested, encoding a polar π-calculus term into the κ-calculus is very simple, and has been done by [Zhang01B]. However, so far we have not found any straightforward technique for the opposite direction. In fact, the polar π-calculus can be considered as a sub-calculus of the κ-calculus.

2.2 The semantics of the κ-calculus

The structural equivalences and labelled transitions in the κ-calculus are shown in Figure 2-2 and Figure 2-3. The central idea of the operational semantics in this calculus is presented by rules tr-IN, tr-CHOI, tr-RELS, tr-SYNC1 and tr-SYNC2. Compare with the π-calculus, we can see that:

1. an input action $\dagger m(\tilde{u})$ invokes a new copy of continuation process P from a *GEC* choice and triggers a lock L which may change Λ, the locking state of *GEC* choice, an unlock signal $\check{\kappa}$ may also change the locking state Λ, but does not change the *GEC* choice context;
2. expressions for different aspects, such as current state, exclusion relation and behaviour of the continuation, can therefore be separated naturally and intuitively.

Summation

str-SUM1: $P_1 \mid \mathbf{0_P} \equiv P_1$; $\quad G_1 \otimes \mathbf{0_G} \equiv G_1$

str-SUM2: $P_1 \mid P_2 \equiv P_2 \mid P_1$; $\quad G_1 \otimes G_2 \equiv G_2 \otimes G_1$

str-SUM3: $P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$; $\quad G_1 \otimes (G_2 \otimes G_3) \equiv (G_1 \otimes G_2) \otimes G_3$

Null

str-NUL: $\Lambda \circ [\mathbf{0_G}] \equiv \mathbf{0_P}$

str-DISJ: $\mathit{addl}(\check{\kappa}@J, \Lambda) \circ [G] \equiv \Lambda \circ [G]$ if $\mathit{guard}(G) \cap J = \varnothing$

Instance

str-INS: $((\tilde{x})P)\langle \tilde{a} \rangle \equiv P\{{}^{\tilde{a}}/_{\tilde{x}}\}$; $\quad ((\tilde{x})G)\langle \tilde{a} \rangle \equiv G\{{}^{\tilde{a}}/_{\tilde{x}}\}$

Scope

str-SCP1: $(\nu\, n)P \equiv P$, if $n \notin \mathit{fn}(P)$; $\quad (\nu\, n)\, G \equiv G$, if $n \notin \mathit{fn}(G)$;

$!(\nu\, \kappa)\beta.P \equiv !\beta.P$, if $\kappa \notin \mathit{fn}(\beta.P)$

str-SCP2: $(\nu\, n_1)(\nu\, n_2)P \equiv (\nu\, n_2)(\nu\, n_1)P$; $\quad (\nu\, n_1)(\nu\, n_2)P \equiv (\nu\, n_1, n_2)P$

str-SCP3: $(\nu\, m)\overline{m}\langle \tilde{y} \rangle \equiv \mathbf{0_P}$; $\quad (\nu\, m)!\dagger m(\tilde{x})L.P \equiv \mathbf{0_G}$;

$(\nu\, \kappa)\hat{\kappa} \equiv \mathbf{0_P}$; $\quad (\nu\, \kappa)\Lambda \circ [G] \equiv (\nu\, \kappa)\Lambda \circ [\mathbf{0_G}]$, if $\mathit{lock}(\mathit{guard}(G), \kappa, \Lambda)$ is true;

$(\nu\, \kappa)\Lambda \circ [G \otimes G'] \equiv \Lambda \circ [G']$, if $\mathit{lock}(\mathit{guard}(G), \kappa, \Lambda)$ and $\kappa \notin \mathit{fn}(G')$

str-SCP4: $\Lambda \circ [(\nu\, \kappa)G] \equiv (\nu\, \kappa)\Lambda \circ [G]$, if $\kappa \notin \mathit{keys}(\Lambda)$;

$(\nu\, n)\, P_1 \mid P_2 \equiv (\nu\, n)(P_1 \mid P_2)$, if $n \notin \mathit{fn}(P_2)$; $\quad (\nu\, n) G_1 \otimes G_2 \equiv (\nu\, n)(G_1 \otimes G_2)$, if $n \notin \mathit{fn}(G_2)$

str-REN: $(\nu\, n_1)\, P \equiv (\nu\, n_2)(P\{{}^{n_2}/_{n_1}\})$, if $n_2 \notin \mathit{fn}(P)$

Figure 2-2 Structural congruence rules for the κ-calculus

tr-OUT: $\dfrac{\bullet}{\overline{m}\langle \tilde{u} \rangle \xrightarrow{\overline{m}\langle \tilde{u} \rangle} \mathbf{0_P}}$, $\quad \dfrac{P \xrightarrow{\overline{m}\langle \tilde{u} \rangle} P', \quad m \notin \tilde{v}}{(\nu\, \tilde{v})P \xrightarrow{(\nu\, \tilde{v})\overline{m}\langle \tilde{u} \rangle} P'}$,

tr-SIG: $\dfrac{\bullet}{\hat{\kappa} \xrightarrow{\hat{\kappa}} \mathbf{0_P}}$

tr-IN: $\dfrac{\Lambda \xrightarrow{\dagger m \uparrow L} \Lambda'}{\Lambda \circ [!(\nu \kappa)\dagger m(\tilde{x})L.P] \xrightarrow{\dagger m(\tilde{u})} (\nu \kappa)(P\{{}^{\tilde{u}}/_{\tilde{x}}\} \mid \Lambda' \circ [!(\nu \kappa)\dagger m(\tilde{x})L.P])}$

tr-CHOI: $\dfrac{\Lambda \circ [G_1] \xrightarrow{\alpha} (\nu\, \kappa)(P \mid \Lambda' \circ [G_1], \quad \kappa \notin \mathit{fn}(G_2)}{\Lambda \circ [G_1 \otimes G_2] \xrightarrow{\alpha} (\nu\, \kappa)(P \mid \Lambda' \circ [G_1 \otimes G_2])}$

tr-RELS: $\dfrac{\Lambda \xrightarrow{\check{\kappa}@J} \Lambda', \quad \text{where } J \supseteq \mathit{guard}(G)}{\Lambda \circ [G] \xrightarrow{\check{\kappa}} \Lambda' \circ [G]}$

tr-PARL: $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$

tr-SYNC1: $\dfrac{P \xrightarrow{(\nu \tilde{v})\overline{m}\langle \tilde{u} \rangle} P', \quad Q \xrightarrow{\dagger m(\tilde{u})} Q', \quad \tilde{v} \cap \mathit{fn}(Q) = \varnothing}{(\nu\, m)(P \mid Q) \xrightarrow{\tau} (\nu\, m)(\nu\, \tilde{v})(P' \mid Q')}$

tr-SYNC2: $\dfrac{P \xrightarrow{\hat{\kappa}} P', \quad Q \xrightarrow{\check{\kappa}} Q'}{(\nu\, \kappa)(P \mid Q) \xrightarrow{\tau} (\nu\, \kappa)(P' \mid Q')}$

tr-RES: $\dfrac{P \xrightarrow{\alpha} P', \quad \tilde{n} \cap \mathit{fn}(\alpha) = \varnothing}{(\nu\, \tilde{n})P \xrightarrow{\alpha} (\nu\, \tilde{n})P'}$

tr-STRUC: $\dfrac{P_1' \equiv P_1, \quad P_1 \xrightarrow{\alpha} P_2, \quad P_2 \equiv P_2'}{P_1' \xrightarrow{\alpha} P_2'}$

Figure 2-3 Labelled transition rules for process terms in the κ-calculus

Justification for our calculus is given in Section 5 where we further discuss modelling of composite objects.

As a normal treatment in this literature, throughout this paper the rule str-REN is often applied automatically and implicitly over fresh names to avoid name clash. For example, a name $n_2 \notin \mathit{fn}(P)$ may be picked up automatically so that the process $(\nu\, n_1)(A\langle \tilde{n}_1, \overline{n}_1 \rangle \mid (\nu\, n_2)P_1\{{}^{n_2}/_{n_1}\})$ can be used to replace $(\nu\, n_1)(A\langle \tilde{n}_1, \overline{n}_1 \rangle \mid (\nu\, n_1)P_1)$ without mention.

Remark 2−5: Similar to the polar π-calculus, in the the κ-calculus the τ action is truly internal, that is, neither visible nor interruptible by external observers. Therefore, the name restrictions in rule tr-SYNC1 and tr-SYNC2 are required. Without it, the synchronisation will not be considered as an internal action, but a two steps action, such as $P \mid Q \xrightarrow{(\nu\tilde{v})\overline{m}\langle\tilde{u}\rangle} . \xrightarrow{\overset{+}{m}(\tilde{u})} (\nu\tilde{v})(P' \mid Q')$ or $P \mid Q \xrightarrow{\hat{\kappa}} . \xrightarrow{\check{\kappa}} P' \mid Q'$, where both steps are visible for external observers. This strong requirement on τ actions is necessary for guaranteeing the standard rule $fn(\tau)=bn(\tau)=\emptyset$ ([Amadio96]) valid, and is necessary for preserving τ actions in output polars substitution.

As usual, let ( )* represent that the contents in ( ) repeating zero or many times, then the weak transitions are defined as:

Definition 2−6: $P \overset{\tau}{\Longrightarrow} P'$ iff $P(\xrightarrow{\tau})^{*}P'$; $\quad P \overset{\alpha}{\Longrightarrow} P'$ iff $P \overset{\tau}{\Longrightarrow} . \xrightarrow{\alpha} . \overset{\tau}{\Longrightarrow} P'$, where $\alpha \neq \tau$.

Reduction relation, a familiar concept in this literature, is defined in a non-standard way in the κ-calculus:

Definition 2−7: $P \longrightarrow P'$ iff $(\nu\, m)P \xrightarrow{\tau} (\nu\, m)P'$ for some m; $\quad P \Longrightarrow P'$ iff $(\nu\, m)P \overset{\tau}{\Longrightarrow} (\nu\, m)P'$ for some m.

Clearly, $P \xrightarrow{\tau} P'$ implies $P \longrightarrow P'$, and $P \overset{\tau}{\Longrightarrow} P'$ implies $P \Longrightarrow P'$, and therefore a variant of the rule tr-SYNC1 can be written as: if $P \xrightarrow{(\nu\tilde{v})\overline{m}\langle\tilde{u}\rangle} P'$ and $Q \xrightarrow{\overset{+}{m}(\tilde{u})} Q'$ where $\tilde{v} \cap fn(Q)=\emptyset$, then $P \mid Q \longrightarrow (\nu\tilde{v})(P' \mid Q')$. Beside the reason we have just discussed, the distinguish between internal action and reduction is also necessary for the new bisimulation relation, and we will find out later.

Definition 2−8: The strong commitments are defined as:

Process P can *commit* the action α, denoted as $P{\downarrow}\alpha$, if there exists some P' such that $P \xrightarrow{\alpha} P'$.
Process P can *commit* on input polar $\overset{+}{m}$, denoted as $P{\downarrow}\overset{+}{m}$, if there exists some input action $\alpha=\overset{+}{m}(\tilde{u})$ s.t. $P{\downarrow}\alpha$;
Process P can *commit* on output polar $\overline{m}$, denoted as $P{\downarrow}\overline{m}$, if there is some output action $\alpha=(\nu\,\tilde{v})\overline{m}\langle\tilde{u}\rangle$ s.t. $P{\downarrow}\alpha$;
Process P can *commit* the action sequence ℓ, denoted as $P{\downarrow}\ell$, if $P \xrightarrow{\alpha_1} . \xrightarrow{\alpha_2} \ldots . \xrightarrow{\alpha_n} P'$, or as an abbreviation, $P \xrightarrow{\ell} P'$;

The weak commitments $\Downarrow$, is obtained by replacing $\longrightarrow$ with $\Longrightarrow$ and $\downarrow$ with $\Downarrow$ though out.

Definition 2−9: Process P' is a *derivative* of process P, if there exists some finite sequence ℓ such that $P \xrightarrow{\ell} P'$.

3 Responsive bisimulation in the κ-calculus

In object-oriented systems, the lock/unlock actions are usually internal activities of objects, and therefore may not be visible from outside. However, while study on a component process of a system or object, these activities have to be observed. In the κ-calculus, the distinction between names for locking keys and for communication allows us to take two different positions in observing processes interactive behaviours:

1. ignore all locking/releasing actions, and adopted the same set bisimulation relations developed in the polar π-calculus;
2. take locking/releasing actions into account and therefore produce the "κ-variation", an even finer version, for each of those bisimulation relations.

Thus, variations of bisimulation relations will be doubled. For every those bisimulations, each κ-version bisimulation is a subset of its non-κ-version counterpart. And in the polar π-calculus, which is a sub-calculus of the κ-calculus, the κ-version and non-κ-version bisimulations will coincide respectively.

Generally say, the κ-version bisimulations are needed for measuring properties of object components, when non-κ-version bisimulations are intersted in measuring overal behaviour of composd objects.

The barbed bisimulation ([Milner92b],[Sangiorgi92b]) is a rather weak relation, which traces the state changes of a process during the course of reductions, and observes which channels available for communication. As a polarised process calculus, in the κ-calculus only output polars (of both communication channels and locking keys) are considered as observable, therefore we adopt a version of barbed bisimulation similar to that in [Zhang01A] for the polar π-calculus.

Definition 3−10 (**barbed bisimulation**): A symmetric relation $\mathcal{S}$ on P-terms is a (strong) barbed bisimulation if whenever $P\mathcal{S}Q$ then $P{\downarrow}a$ implies $Q{\downarrow}a$ for all $a\in\bar{\mathcal{M}}$, and $P\longrightarrow P'$ implies $\exists Q'$ such that $Q\longrightarrow Q'$ and $P'\mathcal{S}Q'$.

Let $\sim_{\mathrm{b}}$ be the largest strong barbed bisimulation. The notion of weak barbed bisimulation $\approx_{\mathrm{b}}$ is obtained by replacing everywhere the transition $\downarrow$ with $\Downarrow$, and $\longrightarrow$ with $\Longrightarrow$ throughout.

For κ-versions, the strong and weak **barbed κ-bisimulation** $\sim_{\kappa\mathrm{b}}$ and $\approx_{\kappa\mathrm{b}}$ respectively, are obtained be extend $a\in\bar{\mathcal{M}}\cup{}^{+}\mathcal{K}\cup\bar{\mathcal{K}}$ in the above definition.

ince barbed bisimulation cannot identify what messages being communicated, it is too rough to measure process's behaviour. Better measurements are needed.

Definition 3−11: In the κ-calculus, **process context** $\mathcal{C}[.]$ is given by $\mathcal{C}::=[.]\,\big|\,(\nu\,\tilde{n})\mathcal{C}\,\big|\,\mathcal{C}|P\,\big|\,\Lambda\circ\lceil!(\nu\kappa)\beta.\,\mathcal{C}\otimes G\rfloor$.

Definition 3−12: Let $\mathcal{C}[.]$ be process context, then we define the barbed equivalences and their κ-versions as
strong and weak **barbed equivalence**: $P\simeq_{\mathrm{b}}Q$ if $\forall\mathcal{C}[.].(\mathcal{C}[P]\sim_{\mathrm{b}}\mathcal{C}[Q])$; $P\cong_{\mathrm{b}}Q$ if $\forall\mathcal{C}[Q].(\mathcal{C}[P]\approx_{\mathrm{b}}\mathcal{C}[Q])$;
strong and weak **barbed κ-equivalence**: $P\simeq_{\kappa\mathrm{b}}Q$ if $\forall\mathcal{C}[.].(\mathcal{C}[P]\sim_{\kappa\mathrm{b}}\mathcal{C}[Q])$; $P\cong_{\kappa\mathrm{b}}Q$ if $\forall\mathcal{C}[Q].(\mathcal{C}[P]\approx_{\kappa\mathrm{b}}\mathcal{C}[Q])$.

Or, for the still weaker versions similar in [Amadio96], let R be arbitrary process, then we define that
strong and weak **barbed** 1-**equivalence**: $P\simeq_{\mathrm{b}1}Q$ if $\forall R.(R|P\sim_{\mathrm{b}}R|Q)$; $P\cong_{\mathrm{b}1}Q$ if $\forall R.(R|P\approx_{\mathrm{b}}R|P)$;
strong and weak **barbed** κ1-**equivalence**: $P\simeq_{\kappa\mathrm{b}1}Q$ if $\forall R.(R|P\sim_{\kappa\mathrm{b}}R|Q)$; $P\cong_{\kappa\mathrm{b}1}Q$ if $\forall R.(R|P\approx_{\kappa\mathrm{b}}R|P)$;

As pointed out by [Zhang01A], weak barbed equivalence is too strong for compositional objects, as illustrated by the example in Figure 1-1, where O_1 and O_2, the two different versions of the same object component, can be expressed in the κ-calculus as $O_1\stackrel{\mathrm{def}}{=}(\nu\,n)\,(!\overset{+}{m}(\tilde{x}).\bar{n}\langle\tilde{x}\rangle\,|\,\lfloor\check{\kappa}@[\overset{+}{n}]\rfloor\circ\lceil!\overset{+}{n}(\tilde{x})L.Body\rfloor)$ and $O_2\stackrel{\mathrm{def}}{=}\lfloor\check{\kappa}@[\overset{+}{m}]\rfloor\circ\lceil!\overset{+}{m}(\tilde{x})L.Body\rfloor$ be two different versions of the same object component. If only output actions are detectable, then within an environment where the input polar of the same channel m is not used elsewhere, the behaviour of O_1 and O_2 can be considered as the same by an external observer. But this similarity of the observation behaviours cannot be captured by the weak barbed equivalence, nor even barbed 1-equivalence. The weak barbed equivalences fail in at least two ways:

$$\begin{tikzcd}[column sep=small]
 & O_1|R \arrow[dl, "\tau"'] \arrow[dr, "\tau"] & & O_2|R \arrow[d, "\tau"] \\
O_1|\bar{p}\langle q\rangle & & Q_1|\overset{+}{m}(x).\bar{x}\langle q\rangle & O_2|\bar{p}\langle q\rangle
\end{tikzcd}$$

Figure 3-1

First, they cannot distinguish between a message sent out from the target process and a message sent to the target process by another agent but buffered in the environment. For example, given the message $\bar{m}\langle\tilde{p}\rangle$, then we have

$$O_1|\bar{m}\langle\tilde{p}\rangle\Longrightarrow Q_1 \text{ and } O_2|\bar{m}\langle\tilde{p}\rangle\Longrightarrow Q_2, \text{ where } Q_1\stackrel{\mathrm{def}}{=}(\nu\,n)(!\overset{+}{m}(\tilde{x}).\bar{n}\langle\tilde{x}\rangle\,|\,\lfloor\check{\kappa}@\{n\}\rfloor\circ\lceil!\overset{+}{n}(\tilde{x})L.Body\rfloor\,|\,\bar{n}\langle\tilde{p}\rangle) \text{ and } Q_2\stackrel{\mathrm{def}}{=}O_2|\bar{m}\langle\tilde{p}\rangle.$$

Since $Q_1\not\Downarrow\bar{m}$ while $Q_2\Downarrow\bar{m}$, therefore $O_1|\bar{m}\langle\tilde{p}\rangle\not\approx_{\mathrm{b}}O_2|\bar{m}\langle\tilde{p}\rangle$, that is, $O_1\not\cong_{\mathrm{b}1}O_2$.

Second, it cannot prevent input names clash between the testing environment and the processes being tested. For example, let $R\stackrel{\mathrm{def}}{=}\overset{+}{m}(\tilde{x}).\bar{x}\langle q\rangle\,|\,\bar{m}\langle\tilde{p}\rangle$, then as shown in Figure 3-1, $O_1|R$ can take two different reduction paths:

$$\text{either } O_1|R\Longrightarrow(\nu\,n)\,(!\overset{+}{m}(\tilde{x}).\bar{n}\langle\tilde{x}\rangle\,|\,\lfloor\check{\kappa}@[\overset{+}{n}]\rfloor\circ\lceil!\overset{+}{n}(\tilde{x})L.Body\rfloor\,|\,\bar{n}\langle\tilde{p}\rangle)\,|\,\overset{+}{m}(\tilde{x}).\bar{x}\langle q\rangle \text{ or } O_1|R\Longrightarrow O_1|\bar{p}\langle q\rangle,$$

while $O_2|R$ has only one reduction path, $O_2|R\Longrightarrow O_2|\bar{p}\langle q\rangle$. Therefore $O_1|R\not\approx_{\mathrm{b}}O_2|R$, that is $O_1\not\cong_{\mathrm{b}1}O_2$.

Another failure in the strong version is, the barbed bisimulation treats synchronisation actions occurred in public channels as single step reduction, and therefore dis-matches them with uncompleted synchronisions which have delay on inputing side.

We need a different technique to measure the observation behaviours, weak enough to ignore the unrelated information and strong enough to distinguish the similarity in responses perceived by outsiders. As with barbed bisimulation, we must note that the state changes of a process caused by internal actions, and we must also be able to detect which communication channels are available for output in all evolved states. What is more, in order to distinguish states, we need to be able to observe what each of the messages output by the process is. The οτ-bisimulation, defined in the same way as that in [Zhang01A], can provide this degree of observation:

Definition 3−13: The (strong) *οτ*-**bisimulation** is a symmetric relation $\mathcal{S}$ on processes such that whenever $P\mathcal{S}Q$ then $P\xrightarrow{\alpha}P'$ implies $Q\xrightarrow{\alpha}Q'$ and $P'\mathcal{S}\,Q'$ for all action α in the form of either $\alpha=(\nu\,\tilde{v})\bar{m}\langle\tilde{u}\rangle$ or $\alpha=\tau$, and $bn(\alpha)\cap fn(Q)=\emptyset$.

The κ-version, (strong) *κοτ*-**bisimulation**, is a strong οτ-bisimulation $\mathcal{S}$ such that whenever $P\mathcal{S}Q$ then $P\xrightarrow{\alpha}P'$ implies $Q\xrightarrow{\alpha}Q'$ and $P'\mathcal{S}\,Q'$ for all $\alpha\in{}^{+}\mathcal{K}\cup{}^{-}\mathcal{K}$.

The weak οτ-bisimulation and weak κοτ-bisimulation are obtained by replacing $\xrightarrow{\alpha}$ with $\xRightarrow{\alpha}$ everywhere above respectively. We denote $\sim_{o\tau}$ be the largest οτ-bisimulation, and $\approx_{o\tau}$ be the largest weak οτ-bisimulation, $\sim_{\kappa o\tau}$ be the largest κοτ-bisimulation, and $\approx_{\kappa o\tau}$ be the largest weak κοτ-bisimulation.

Lemma 3−14: Each of κ-version and non-κ-version οτ-bisimulations, $\mathcal{S}$, is preserved by restriction, that is, $P\mathcal{S}Q$ implies $(\nu\,\tilde{n})P\mathcal{S}(\nu\,\tilde{n})Q$.

Proof: This can be proven by show that $\mathcal{R}\stackrel{\text{def}}{=}\{((\nu\,\tilde{n})P,(\nu\,\tilde{n})Q): P\mathcal{S}Q\}$ is a $\mathcal{S}$. Here we only give the proof for the strong κ-version, $\mathcal{S}\subseteq\sim_{\kappa o\tau}$, all others can be proven similarly. Assume $(\nu\,\tilde{n})P\xrightarrow{\alpha}P'$ for some arbitrary action α, where α is not a communiction input action (i.e., $\alpha\neq{}^{\dagger}\bar{m}(\bar{v})$), then it is only possible in one of the following two cases:

1. $\tilde{n}\cap fn(\alpha)=\emptyset$ and $P\xrightarrow{\alpha}P''$. By rule tr-RES, $(\nu\,\tilde{n})P\xrightarrow{\alpha}(\nu\,\tilde{n})P''$, so $P'\equiv(\nu\,\tilde{n})P''$. By $P\mathcal{S}Q$, we have $Q\xrightarrow{\alpha}Q''$ and $P''\mathcal{S}Q''$. By tr-RES, $(\nu\,\tilde{n})Q\xrightarrow{\alpha}(\nu\,\tilde{n})Q''$, and we have $((\nu\,\tilde{n})P'',(\nu\,\tilde{n})Q'')\in\mathcal{R}$;
2. α is an output action of the form $\alpha=(\nu\,\tilde{v})\bar{m}\langle\tilde{u}\rangle$ where $m\notin\tilde{n}$ and $\tilde{v}_1=\tilde{n}\cap(\tilde{u}-\tilde{v})\neq\emptyset$, and $P\xrightarrow{\bar{m}\langle\tilde{u}\rangle}P''$. By $P\mathcal{S}Q$, we have $Q\xrightarrow{\bar{m}\langle\tilde{u}\rangle}Q''$ and $P''\mathcal{S}Q''$. Let $\tilde{v}_2=\tilde{n}-\tilde{v}_1$, by rule str-SCP2 and str-SUM2, $(\nu\,\tilde{n})P\equiv(\nu\,\tilde{v}_1)(\nu\,\tilde{v}_2)P$ and $(\nu\,\tilde{n})Q\equiv(\nu\,\tilde{v}_1)(\nu\,\tilde{v}_2)Q$. By the tr-OUT, we got $(\nu\,\tilde{v}_1)(\nu\,\tilde{v}_2)P\xrightarrow{\alpha}(\nu\,\tilde{v}_2)P''$ and $(\nu\,\tilde{v}_1)(\nu\,\tilde{v}_2)Q\xrightarrow{\alpha}(\nu\,\tilde{v}_2)Q''$, however $((\nu\tilde{v}_2)P'',(\nu\tilde{v}_2)Q'')\in\mathcal{R}$.

By the definition of κοτ-bisimulation $\mathcal{S}$, we have $\mathcal{R}\subseteq\mathcal{S}$. ■

The οτ-bisimulation gives a measurement on processes' states by observing available reductions and output actions, but can not determine how a process responses to incoming messages, since communicating input actions are not observed. To determine responsive behaviours, we introduce a new term for specifying input messages.

Notation 3−15: We add the auxiliary *P*-term $[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P$, the *localisation* of the sent message $\bar{m}\langle\tilde{u}\rangle$ with process *P*, into the process syntax. Properties for this term are shown in Figure 3-2.

Structural equivalence :

lStr_NULL $[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]\,\mathbf{0}\equiv\mathbf{0}$;

lStr_IND $([{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P)\,|\,Q\equiv[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle](P\,|\,Q)$, if $m\notin fin(Q)$;

lStr_LOC $(\nu\,m)\,[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P\equiv(\nu\,m)\,(\bar{m}\langle\tilde{u}\rangle\,|\,P)$;

lStr_SUM2′ $[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle][{}^{\dagger}\bar{n}\langle\tilde{v}\rangle]P\equiv[{}^{\dagger}\bar{n}\langle\tilde{v}\rangle][{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P$;

Transition :

lTr_SYNC3 $\dfrac{P\xrightarrow{{}^{\dagger}m(\tilde{u})}P'}{[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P\xrightarrow{\tau}P'}$;

lTr_INV $\dfrac{P\xrightarrow{\alpha}P'\qquad \alpha\neq{}^{\dagger}m(\tilde{u})}{[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P\xrightarrow{\alpha}[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P'}$.

Figure 3-2 Localised output action.

The term $[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P$ couples P with the message $\tilde{u}$ which is buffered in channel, and unobservable from outside, even though the output polar $\bar{m}$ may have been known by outsiders. We may consider the difference between $\bar{m}\langle\tilde{u}\rangle\,|\,P$ and $[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]P$ as that, in the former the $\bar{m}\langle\tilde{u}\rangle$ is an outging message to be bufered into the channel *m*, while in the latter, $[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]$ is a buffered message arriving from the channel *m* and waiting to be picked up by *P*. The $[{}^{\dagger}\bar{m}\langle\tilde{u}\rangle]$ privatises neither polar ${}^{\dagger}m$ nor $\bar{m}$, but the message $\tilde{u}$. In other words, the $[{}^{\dagger}m\langle\tilde{u}\rangle]$ is like the mailbox on the right side in , with the message $\tilde{u}$ in it, and only P or its derivatives may (but not have to) consume this message. That is the reason why the input polar ${}^{\dagger}m$ rather than output polar $\bar{m}$ appears in $[{}^{\dagger}m\langle\tilde{u}\rangle]P$.

The term $[{}^{\dagger}m\langle\tilde{u}\rangle]P$ is not for modelling processes, but only designed to express r1-bisimulation relations between processes. In this sense, we may read $[{}^{\dagger}m\langle\tilde{u}\rangle]P$ as "the behaviour of the black box P while provided with the test message $\tilde{u}$ via channel m", and this behaviour depends on whether and when P or its derivatives able to access the input port ${}^{\dagger}m$. From this point of view, the using of input polar ${}^{\dagger}m$ rather than output polar $\bar{m}$ is necessary to prevent an input polar substitution, caused by input prefixing, changes the static behaviour of P.

The rule lTr_SYNC3 added a new case for defining the τ action. Unlike in rule tr-SYNC1, here is no name restriction is required. However, since only the input polar, ${}^{\dagger}m$, of the channel name m is involved, and the reservation of τ actions is maintained input prefixing.

Corollary 3−16: The following conclusion can be immediately drew from the rules in Figure 3-2:
(1) If $P \xrightarrow{\bar{m}\langle\tilde{u}\rangle} P'$ then $(\nu\, m)P \equiv (\nu\, m)\,[{}^{\dagger}m\langle\tilde{u}\rangle]P'$;
(2) $P\not\downarrow\alpha$ implies $(\,[{}^{\dagger}m\langle\tilde{u}\rangle]P\,)\not\downarrow\alpha$ if $\alpha\neq\tau$, or, $\alpha=\tau$ but $P\not\downarrow{}^{\dagger}m$;
(3) $P\downarrow{}^{\dagger}m$ implies $(\,[{}^{\dagger}m\langle\tilde{u}\rangle]P\,)\downarrow\tau$;
(4) $(\,[{}^{\dagger}m\langle\tilde{u}\rangle]P\,)\not\downarrow{}^{\dagger}m(\tilde{u})$.

Now we can begin to introduce new behaviour equivalence relations.

Definition 3−17: Let $\mathcal{T}[.]$ be the **responsive testing context** of syntax $\mathcal{T} ::= [.] \mid [{}^{\dagger}m\langle\tilde{u}\rangle]\mathcal{T}$, then we define
the strong and weak **responsive equivalence**: $P\simeq_r Q$ iff $\forall\mathcal{T}.(\mathcal{T}[P]\sim_{o\tau}\mathcal{T}[Q])$, $P\cong_r Q$ iff $\forall\mathcal{T}.(\mathcal{T}[P]\approx_{o\tau}\mathcal{T}[Q])$;
the strong and weak ***κ*r- equivalence**: $P\simeq_{\kappa r} Q$ iff $\forall\mathcal{T}.(\mathcal{T}[P]\sim_{\kappa o\tau}\mathcal{T}[Q])$, $P\cong_{\kappa r} Q$ iff $\forall\mathcal{T}.(\mathcal{T}[P]\approx_{\kappa o\tau}\mathcal{T}[Q])$.

This definition gives a quite clear description about the meaning of equivalence in responsive behaviour, but is not so useful since it requires the exhaustive testing over the infinite set of responsive testing contexts. A more practical definition is the r1-bisimulation, named so because the structurally comparable to the 1-bisimulation in [Amadio96].

Definition 3−18: The strong (or weak) *r*1**-bisimulation** is a strong (or weak, respectively) oτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then $[{}^{\dagger}m\langle\tilde{u}\rangle]P\,\mathcal{S}\,[{}^{\dagger}m\langle\tilde{u}\rangle]Q$ for all $[{}^{\dagger}m\langle\tilde{u}\rangle]$.

We denote the largest strong r1-bisimulation as $\sim_{r1}$, and the largest weak r1-bisimulation as $\approx_{r1}$.

The κ-versions, strong and weak *κr*1**-bisimulation** $\sim_{\kappa r1}$ and $\approx_{\kappa r1}$, are defined by replacing oτ-bisimulation with its κ-version, the κoτ-bisimulation, in the above definition.

Lemma 3−19: The responsive equivalence and r1-bisimulation are coincide for both κ-version and non-κ-version, i.e., $\sim_{\kappa r1}\equiv\simeq_{\kappa r}$, $\approx_{\kappa r1}\equiv\cong_{\kappa r}$, $\sim_{r1}\equiv\simeq_r$ and $\approx_{r1}\equiv\cong_r$.

Proof: Proven by induction. We only show $\sim_{\kappa r1}\equiv\simeq_{\kappa r}$ here, and all other cases can be proven in similar way.

$\sim_{\kappa r1}\subseteq\simeq_{\kappa r}$: Let $P\sim_{\kappa r1}Q$, then we can write $\mathcal{T}_0[P]\sim_{\kappa r1}\mathcal{T}_0[Q]$ where $\mathcal{T}_0 \stackrel{\text{def}}{=} [.]$.
Assume $\mathcal{T}_i[P]\sim_{\kappa r1}\mathcal{T}_i[Q]$ is held for some responsive testing context $\mathcal{T}_i$.
By the definition of $\sim_{\kappa r1}$, for all $[{}^{\dagger}m\langle\tilde{u}\rangle]$, we have $\mathcal{T}_{i1}[P]\sim_{\kappa r1}\mathcal{T}_{i1}[Q]$ for each of $\mathcal{T}_{i1}\stackrel{\text{def}}{=}[{}^{\dagger}m\langle\tilde{u}\rangle]\mathcal{T}_i$.
By induction, and notice $\sim_{\kappa r1}\subseteq\sim_{\kappa o\tau}$ from the definition of $\sim_{\kappa r1}$, we conclude that,
$P\sim_{\kappa r1}Q$ implies $\forall\mathcal{T}.(\mathcal{T}[P]\sim_{\kappa o\tau}\mathcal{T}[Q])$, that is, $P\simeq_{\kappa r}Q$, by the definition of $\simeq_{\kappa r}$.

$\sim_{\kappa r1}\supseteq\simeq_{\kappa r}$: Let $P\simeq_{\kappa r}Q$, then it implies $P\sim_{\kappa o\tau}Q$ because $\mathcal{T}_0[P]\sim_{\kappa o\tau}\mathcal{T}_0[Q]$ for $\mathcal{T}_0\stackrel{\text{def}}{=}[.]$, that is, $\simeq_{\kappa r}\subseteq\sim_{\kappa o\tau}$.
It also implies $[{}^{\dagger}m\langle\tilde{u}\rangle]P\sim_{\kappa o\tau}[{}^{\dagger}m\langle\tilde{u}\rangle]Q$ for all $[{}^{\dagger}m\langle\tilde{u}\rangle]$, because $\mathcal{T}_1[P]\sim_{\kappa o\tau}\mathcal{T}_1[Q]$ for each $\mathcal{T}_1\stackrel{\text{def}}{=}[{}^{\dagger}m\langle\tilde{u}\rangle][.]$.
And still, it implies for each responsive testing context $\mathcal{T}$, if we write $\mathcal{T}_2\stackrel{\text{def}}{=}\mathcal{T}[\,\mathcal{T}_1[.]\,]$ then $\mathcal{T}_2[P]\sim_{\kappa o\tau}\mathcal{T}_2[Q]$,
In other words, $\forall\mathcal{T}.(\mathcal{T}[[{}^{\dagger}m\langle\tilde{u}\rangle]P]\sim_{\kappa o\tau}\mathcal{T}[[{}^{\dagger}m\langle\tilde{u}\rangle]Q])$. That is, $P\simeq_{\kappa r}Q$ implies $[{}^{\dagger}m\langle\tilde{u}\rangle]P\simeq_{\kappa r}[{}^{\dagger}m\langle\tilde{u}\rangle]Q$ for all $[{}^{\dagger}m\langle\tilde{u}\rangle]$. Therefore $\simeq_{\kappa r}\subseteq\sim_{\kappa r1}$ by the definition of $\sim_{\kappa r1}$. ■

It is easy to verify that $O_1 \approx_{r1} O_2$ and $O_1 \approx_{\kappa r1} O_2$ hold for the processes O_1 and O_2 mentioned in the example at earlier of this session. The r1-bisimulation providers a test platform for measureing behavioural equivalence from outside of target processes.

However, while responsive equivalences and r1-bisimulations provide a good base for describing similarities of responsive behaviours, they tell little about why or when two processes may offer similar behaviours. For closer study, we need an inside view observing input actions.

Definition 3−20 : The (strong) **responsive bisimulation** is a (strong) οτ-bisimulation $\mathcal{S}$ such that whenever $P\mathcal{S}Q$ then $P \xrightarrow{\dagger m(\tilde{u})} P'$ implies either $Q \xrightarrow{\dagger m(\tilde{u})} Q'$ and $P'\mathcal{S}Q'$, or $Q \xrightarrow{\tau} Q'$ and $P'\mathcal{S}[\dagger m\langle\tilde{u}\rangle]Q'$.

The weak responsive bisimulation is obtained by replacing transitions with weak transitions everywhere. We denote $\sim_r$ and $\approx_r$ be the largest strong and weak responsive bisimulation respectively. Clearly, $\sim_r \subseteq \approx_r$.

The κ-versions, strong and weak *κr*-**bisimulation** $\sim_{\kappa r}$ and $\approx_{\kappa r}$, are defined by replace οτ-bisimulation with κοτ-bisimulation in the above definitions. Clearly, $\sim_{\kappa r} \subseteq \approx_{\kappa r}$.

Lemma 3−21: The responsive bisimulation and r1-bisimulation are coincide for both κ-version and non-κ-version, i.e., $\sim_{\kappa r} \equiv \sim_{\kappa r1}$, $\approx_{\kappa r} \equiv \approx_{\kappa r1}$, $\sim_r \equiv \sim_{r1}$ and $\approx_r \equiv \approx_{r1}$.

Proof: Here we only show that for $\sim_{\kappa r} \equiv \sim_{\kappa r1}$, and other cases can be proven in a similar way.

$\sim_{\kappa r} \subseteq \sim_{\kappa r1}$: Let $\mathcal{R} \stackrel{\text{def}}{\equiv} \{([\dagger m\langle\tilde{u}\rangle]P, [\dagger m\langle\tilde{u}\rangle]Q) : P\mathcal{S}Q\} \cup \mathcal{S}$ for $\mathcal{S} = \sim_{\kappa r}$. Assume $[\dagger m\langle\tilde{u}\rangle]P \xrightarrow{\alpha} P'$ for some an arbitrary action α, by the rules in Figure 3-2 and by Corollary 3-16 (4), it is only possible in the following two cases:

(1) $P \xrightarrow{\alpha} P''$ and $\alpha \neq \dagger m(\tilde{u})$, then $P' = [\dagger m\langle\tilde{u}\rangle]P''$. Since $P \sim_{\kappa r} Q$, we have
either $Q \xrightarrow{\alpha} Q''$, $P'' \sim_{\kappa r} Q''$ and $[\dagger m\langle\tilde{u}\rangle]Q \xrightarrow{\alpha} [\dagger m\langle\tilde{u}\rangle]Q''$, and therefore $([\dagger m\langle\tilde{u}\rangle]P'', [\dagger m\langle\tilde{u}\rangle]Q'') \in \mathcal{R}$;
or $\alpha = \dagger n(\tilde{v})$, $Q \xrightarrow{\tau} Q''$, $P'' \sim_{\kappa r} [\dagger n\langle\tilde{v}\rangle]Q''$, that is, $[\dagger m\langle\tilde{u}\rangle]P \xrightarrow{\dagger n(\tilde{v})} [\dagger m\langle\tilde{u}\rangle]P''$ and $[\dagger m\langle\tilde{u}\rangle]Q \xrightarrow{\tau} [\dagger m\langle\tilde{u}\rangle]Q''$. By rule lStr_SUM2′, $[\dagger n\langle\tilde{v}\rangle][\dagger m\langle\tilde{u}\rangle]Q'' \equiv [\dagger m\langle\tilde{u}\rangle][\dagger n\langle\tilde{v}\rangle]Q''$, therefore $([\dagger m\langle\tilde{u}\rangle]P'', [\dagger n\langle\tilde{v}\rangle][\dagger m\langle\tilde{u}\rangle]Q'') \in \mathcal{R}$;

(2) $\alpha = \tau$ and $P \xrightarrow{\dagger m(\tilde{u})} P'$, then by $P \sim_{\kappa r} Q$ it implies
either $Q \xrightarrow{\dagger m(\tilde{u})} Q'$, $[\dagger m\langle\tilde{u}\rangle]Q \xrightarrow{\tau} Q'$ and $P' \sim_{\kappa r} Q'$, that is $(P', Q') \in \mathcal{R}$ since $\mathcal{R} \supseteq \sim_{\kappa r}$;
or $Q \not\downarrow \dagger m(\tilde{u})$, then $Q \xrightarrow{\tau} Q'$ and $P' \sim_r [\dagger m\langle\tilde{u}\rangle]Q'$, that is $(P', [\dagger m\langle\tilde{u}\rangle]Q') \in \mathcal{R}$ since $\mathcal{R} \supseteq \sim_{\kappa r}$;

Then by definition of $\sim_{\kappa r}$, we have $\mathcal{R} \subseteq \sim_{\kappa r}$, that is, $P \sim_{\kappa r} Q$ implies $[\dagger m\langle\tilde{u}\rangle]P \sim_{\kappa r} [\dagger m\langle\tilde{u}\rangle]Q$. Because $\sim_{\kappa r} \subseteq \sim_{\kappa o\tau}$, and because $[\dagger m\langle\tilde{u}\rangle]$ is arbitrary here, we have $\sim_{\kappa r} \subseteq \sim_{\kappa r1}$ by the definition of $\sim_{\kappa r1}$.

$\sim_{\kappa r} \supseteq \sim_{\kappa r1}$: Let $P \sim_{\kappa r1} Q$, then $[\dagger m\langle\tilde{u}\rangle]P \sim_{\kappa r1} [\dagger m\langle\tilde{u}\rangle]Q$ for all $[\dagger m\langle\tilde{u}\rangle]$. Assume $P \xrightarrow{\alpha} P'$ for some action α:

If α is a non-input action, then $Q \xrightarrow{\alpha} Q'$ and $P' \sim_{\kappa r1} Q'$;

If $\alpha = \dagger m(\tilde{u})$, then $[\dagger m\langle\tilde{u}\rangle]P \xrightarrow{\tau} P'$. By $[\dagger m\langle\tilde{u}\rangle]P \sim_{\kappa r1} [\dagger m\langle\tilde{u}\rangle]Q$ and $[\dagger m\langle\tilde{u}\rangle]Q \xrightarrow{\tau} Q''$, it must be
either $Q \xrightarrow{\dagger m(\tilde{u})} Q'$ and $Q'' \equiv Q'$. But $P' \sim_{\kappa o\tau} Q'$ since $\sim_{\kappa r1} \subseteq \sim_{\kappa o\tau}$. Let $\mathcal{R}_1 = \{(P,Q), (P',Q')\}$, then $\mathcal{R}_1 \subseteq \sim_{\kappa o\tau}$;
or $Q \xrightarrow{\tau} Q'$ and $Q'' \equiv [\dagger m\langle\tilde{u}\rangle]Q'$. But $P' \sim_{\kappa o\tau} [\dagger m\langle\tilde{u}\rangle]Q'$ since $\sim_{\kappa r1} \subseteq \sim_{\kappa o\tau}$. Let $\mathcal{R}_2 = \{(P,Q), (P', [\dagger m\langle\tilde{u}\rangle]Q')\}$, then $\mathcal{R}_2 \subseteq \sim_{\kappa o\tau}$;

Let $\mathcal{R} = \sim_{\kappa r1} \cup \mathcal{R}_1 \cup \mathcal{R}_2$, then $\mathcal{R} \subseteq \sim_{\kappa o\tau}$ since $\sim_{\kappa r1} \subseteq \sim_{\kappa o\tau}$, then we have $\mathcal{R} \subseteq \sim_{\kappa r}$ by the definition of $\sim_{\kappa r}$. ■

Corollary 3−22: The responsive bisimulation and responsive equivalence are coincide for both κ-version and non-κ-version, i.e., $\sim_{\kappa r} \equiv \simeq_{\kappa r}$, $\approx_{\kappa r} \equiv \approxeq_{\kappa r}$, $\sim_r \equiv \simeq_r$ and $\approx_r \equiv \approxeq_r$.

4 Properties of the responsive bisimulation

In this section we explore some formal properties of our newly defined responsive bisimulation and establish connection with some conventional bisimulations, which include, their preservability in parallel composition, name substitution and *GEC* choice, their congruency for autonomous processes.

Corollary 4−23: The responsive bisimulations are preserved by localisation. That is, let $\mathcal{S}$ be any of $\sim_r$, $\approx_r$, $\sim_{\kappa r}$ or $\approx_{\kappa r}$, then $P\mathcal{S}Q$ implies $[\dagger m\langle\tilde{u}\rangle]P\,\mathcal{S}\,[\dagger m\langle\tilde{u}\rangle]Q$ for all $[\dagger m\langle\tilde{u}\rangle]$.

Proof: Let $\mathcal{R}$ be the corresponding r1-bisimulation ($\sim_{r1}$, $\approx_{r1}$, $\sim_{\kappa r1}$ or $\approx_{\kappa r1}$) respect to $\mathcal{S}$, then by Lemma 3-21, $P\mathcal{S}Q$ implies $P\mathcal{R}Q$, which then implies that $[\dagger m\langle\tilde{u}\rangle]P\,\mathcal{R}[\dagger m\langle\tilde{u}\rangle]Q$ for all $[\dagger m\langle\tilde{u}\rangle]$ according to the definition of r1-bisimulation, then again by Lemma 3-21, we have $[\dagger m\langle\tilde{u}\rangle]P\,\mathcal{S}\,[\dagger m\langle\tilde{u}\rangle]Q$. ∎

Lemma 4−24: The responsive bisimulations are equivalences.

Proof: Here we only give the proof for the strong κ-version $\sim_{\kappa r}$, and all other cases can be proven similarly.

Reflexive : $P\sim_{\kappa r}P$ for any P, according to the definition of $\sim_{\kappa r}$;

Symmetric : if $P\sim_{\kappa r}Q$ then $Q\sim_{\kappa r}P$, by the definition of $\sim_{\kappa r}$;

Transitive : Let $P_1\mathcal{R}_1P_2$ and $P_2\mathcal{R}_2P_3$, where $\mathcal{R}_1\subseteq\sim_{\kappa r}$ and $\mathcal{R}_2\subseteq\sim_{\kappa r}$, and therefore $P_1(\mathcal{R}_1\mathcal{R}_2)P_3$. For arbitrary action α, if α is not an input communication act, then

$$P_1\xrightarrow{\alpha}P_1' \text{ implies } P_2\xrightarrow{\alpha}P_2' \text{ and } P_1'\mathcal{R}_1P_2', \text{ which further implies } P_3\xrightarrow{\alpha}P_3' \text{ and } P_2'\mathcal{R}_2P_3', \text{ that is, } P_1'(\mathcal{R}_1\mathcal{R}_2)P_3'.$$

If α is an input communication act, say $\alpha=\dagger m(\tilde{u})$, and $P_1\xrightarrow{\dagger m(\tilde{u})}P_1'$, then we may have either

$P_2\xrightarrow{\dagger m(\tilde{u})}P_2'$ and $P_1'\mathcal{R}_1P_2'$, $P_3\xrightarrow{\dagger m(\tilde{u})}P_3'$ and $P_2'\mathcal{R}_2P_3'$, and therefore $P_1'(\mathcal{R}_1\mathcal{R}_2)P_3'$;

or $P_2\xrightarrow{\dagger m(\tilde{u})}P_2'$ and $P_1'\mathcal{R}_1P_2'$, $P_3\xrightarrow{\tau}P_3'$ and $P_2'\mathcal{R}_2[\bar{m}\langle\tilde{u}\rangle]P_3'$, and therefore $P_1'(\mathcal{R}_1\mathcal{R}_2)[\bar{m}\langle\tilde{u}\rangle]P_3'$;

or $P_2\xrightarrow{\tau}P_2'$ and $P_1'\mathcal{R}_1[\bar{m}\langle\tilde{u}\rangle]P_2'$, $P_3\xrightarrow{\tau}P_3'$ and $P_2'\mathcal{R}_2P_3'$. By $\sim_{\kappa r}\equiv\sim_{\kappa r1}$, we have $\mathcal{R}_2\subseteq\sim_{\kappa r1}$, so $[\bar{m}\langle\tilde{u}\rangle]P_2'\mathcal{R}_2[\bar{m}\langle\tilde{u}\rangle]P_3'$, and therefore $P_1'(\mathcal{R}_1\mathcal{R}_2)[\bar{m}\langle\tilde{u}\rangle]P_3'$. By the definition, $(\mathcal{R}_1\mathcal{R}_2)\subseteq\sim_{\kappa r}$. ∎

There is a problem: the responsive bisimulations are not be preserved by parallel composition in general. For instance, with the O_1 and O_2 of the previous example, we have $O_1\sim_rO_2$, but $(O_1|O_3)\nsim_r(O_2|O_3)$ for $O_3\stackrel{\text{def}}{=}\lfloor\,\rfloor\circ[!\dagger m(\tilde{v})L.R]$, because the occurrence of input polar $\dagger m$ in O_3 has changed the ability of O_1 on receiving message from $\dagger m$. However, as mentioned at the beginning of this paper, the purpose of our study is about object modelling, and as the nature of object systems, the ownership of each input port should be unique. For example, the object identity of an object is uniquely owned by no one else but that object; each method of each object is also uniquely identified so that no message would be delivered to wrong destination. In general, as mentioned in the previous session, each input polar has a static scope (or ownership), and will never appears outside this scope.

When responsive bisimulation is strictly restricted within the problem domain, objects modelling, where the responsive bisimulation is needed, then its preservation in parallel composition can be guaranteed, as shown later.

Definition 4−25: Let $\dagger m$ be the input polar of a communication channel name m, P be a process for which $m\in fin(P)$, and $\mathcal{E}$ be the context $\mathcal{E}[.]\stackrel{\text{def}}{=}(\nu\,\tilde{n})\,(Env\,|\,[.]\,)$ where $m\notin fin(Env)$ while m may or may not be a member of $\tilde{n}$. We say that, P is an *owner* of $\dagger m$ (or say, $\dagger m$ is owned by P) with respect to the *environment Env*;

Env is an environment free of $\dagger m$ (or say, $\dagger m$*-free environment*);

$\mathcal{E}[.]$ is an $\dagger m$*-safe* environment context, or $\dagger m$-safe environment for short.

An $\dagger m$-safe environment only allows the process in the hole to consume a message sent along the channel m, ensuring no interference from the environment. It reflects the fact that the responsive behaviour of a process can be measured only when messages sent to it are guaranteed not to be intercepted by some other process.

Definition 4−26 A process P is *safe* for Env, and the environment Env is said to be *safe* for P, if P is the owner of all $m \in fin(P)$ respect to the environment Env, i.e., $fin(P) \cap fin(Env) = \emptyset$. We may call P an *safe process*, when the behaviour of P is only considered within environments which are safe for P.

A process P is *autonomous* if $fin(P) = \emptyset$.

Lemma 4−27: The process safety is preserved by evolution. That is, if $fin(P) \cap fin(Env) = \emptyset$ holds for processes P and Env, then $fin(P') \cap fin(Env') = \emptyset$ holds for all P' and Env', which are derivatives of P and Env respectively.

Proof: Simply because the input polar of a channel cannot be transmitted by communication. ■

Corollary 4−28: An autonomous process and all its derivatives are safe to any system.

When modelling objects in the κ-calculus, all method bodies can be considered as autonomous, since after parameters passed through the method interface, further input (if any) can only be performed via channels that were initially private and informed to the senders by the forked method body. An object itself is initially autonomous while creation, until its name, the unique identification, is exported to its environment. Its method names can also be considered as initially private to the object, and then exported to the caller during each method call. For example, similar to [Walker95] and [Zhang97] amongst others, the method call `o.m1(a1,a2)` may be modelled as $(\nu\, mset)(\bar{o}\langle \overline{mset}\rangle \mid \ddagger mset(\tilde{m}).\bar{m}_1\langle \bar{a}_1, \bar{a}_2\rangle)$, and on the object side the encoding will look like $(\nu\ \tilde{m})\ (!\ddagger o(\overline{mset}).\ \overline{mset}\langle \tilde{m}\rangle \mid \Lambda \circ [\ \bigotimes !\ddagger m_i(\tilde{v}) L_i.Body_i]\ )$.

Proposition 4−29: The responsive bisimulations are preserved by parallel composition for safe processes. That is, to each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, whenever $P_1 \mathcal{S} P_2$ implies $(P_1 \mid P)\mathcal{S}(P_2 \mid P)$ for all P which satisfying $fin(P) \cap (fin(P_1) \cup fin(P_2)) = \emptyset$.

Proof: Here we only show that for the weak κ-version $\mathcal{S} \subseteq \approx_{\kappa r}$, all other cases can be proven similarly.

Let $\equiv_P$ be the congruence induced by the commutativity and associativity laws for parallel composition "$\mid$" in Figure 2-2 and rule lStr_SUM2′ in Figure 3-2, and let relation $\mathcal{R} \stackrel{def}{=} \{(P_1 \mid P, P_2 \mid P)\colon (P_1 \approx_{\kappa r} P_2) \wedge (fin(P) \cap (fin(P_1 \mid P_2)) = \emptyset)\} \cup \approx_{\kappa r}$. Let $Q_1 \stackrel{def}{=} P_1 \mid P$ and $Q_2 \stackrel{def}{=} P_2 \mid P$, and $Q_1 \stackrel{\alpha}{\Longrightarrow} Q_1'$ for some action α, by $P_1 \approx_{\kappa r} P_2$, it must in one of the following three cases:

(1) $P_1 \stackrel{\alpha}{\Longrightarrow} P_1'$, $P_2 \stackrel{\alpha}{\Longrightarrow} P_2'$ and $P_1' \approx_{\kappa r} P_2'$, and therefore $Q_1' \equiv P_1' \mid P$, $Q_2 \stackrel{\alpha}{\Longrightarrow} Q_2'$ for $Q_2' \stackrel{def}{=} P_2' \mid P$;

(2) $P \stackrel{\alpha}{\Longrightarrow} P'$, and therefore $Q_1' \equiv P_1 \mid P'$ and $Q_2 \stackrel{\alpha}{\Longrightarrow} Q_2'$ for $Q_2' \stackrel{def}{=} P_2 \mid P'$;

(3) $\alpha = \ddagger m(\tilde{u})$, $P_1 \stackrel{\ddagger m(\tilde{u})}{\Longrightarrow} P_1'$, $P_2 \stackrel{\tau}{\Longrightarrow} P_2'$ and $P_1' \approx_r [\ddagger m\langle \tilde{u}\rangle] P_2'$, by the autonomous condition $fin(P) \cap (fin(P_1) \cup fin(P_2)) = \emptyset$, it must be $m \notin fin(P)$ and therefore $[\ddagger m\langle \tilde{u}\rangle] P_2' \mid P \equiv [\ddagger m\langle \tilde{u}\rangle](P_2' \mid P)$, that is, $Q_1' \equiv P_1' \mid P$, and $Q_2 \stackrel{\tau}{\Longrightarrow} Q_2'$ for $Q_2' \stackrel{def}{=} P_2' \mid P$;

The cases with synchronisation, such as $P_1 \stackrel{(\nu\, \tilde{v})\bar{m}\langle \tilde{u}\rangle}{\Longrightarrow} P_1'$ and $P \stackrel{\ddagger m(\tilde{u})}{\Longrightarrow} P'$, or $P_1 \stackrel{\ddagger m(\tilde{u})}{\Longrightarrow} P_1'$ and $P \stackrel{(\nu\, \tilde{v})\bar{m}\langle \tilde{u}\rangle}{\Longrightarrow} P'$, or $P_1 \stackrel{\hat{K}}{\Longrightarrow} P_1'$ and $P \stackrel{\check{K}}{\Longrightarrow} P'$, or $P_1 \stackrel{\check{K}}{\Longrightarrow} P_1'$ and $P \stackrel{\hat{K}}{\Longrightarrow} P'$, have been covered by theses three above cases, according to Remark 2-5, and therefore need not to be considered separately.

Since $(Q_1', Q_2') \in \mathcal{R}$ for cases 1-2, and $(Q_1', [\ddagger m\langle \tilde{u}\rangle] Q_2') \in \mathcal{R} \circ \equiv_P$ for the third case, therefore $\mathcal{R}$ is a $\approx_{\kappa r}$ upto $\equiv_P$. ■

Let σ denote a name substitution of the form $\sigma = \{\tilde{u}/\tilde{x}\}$, which is over output communication polars only, otherwise standard. Whenever applied to a process or an action, bound names (in pairs of both polars) are automatically renamed to avoid conflict. We do not need to consider substitution over input polars nor locking/releasing keys, because they can not be sent through channels in the κ-calculus, and clearly the safeness of processes is preserved by the output polar substitution.

Proposition 4−30: The responsive bisimulations are preserved by output polarity name substitution. That is, to each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, $P \mathcal{S} Q$ implies $P\sigma \mathcal{S} Q\sigma$ for all $\sigma = \{\tilde{u}/\tilde{x}\}$.

Proof: Again these can be proven by showing $\mathcal{R} \stackrel{def}{=} \{(P\sigma, Q\sigma)\colon P \mathcal{S} Q\} \cup \mathcal{S}$ is a $\mathcal{S}$ upto $\equiv_P$. Here we only show that for the strong κ-version, $\mathcal{S} \subseteq \sim_{\kappa r}$, all others can be proven similarly. Lets exam all the possible actions that $P\sigma$ may take:

$P\sigma \xrightarrow{(\nu\, \tilde{s})\bar{m}\langle \tilde{u}\rangle} P''$: it is only possible when there exists some P'' such that $P\sigma \equiv (\nu\, \tilde{s})(\bar{m}\langle \tilde{u}\rangle \mid P')$, by the transition rules listed in Figure 2-3 and Figure 3-2. Remember the implicity renaming over fresh names to avoid name clash, and notice that the substitution only effects to free output polars, then there must exist some $\bar{c}$, $\tilde{v}$ and P' such that $\bar{m}=\bar{c}\sigma$, $\tilde{u}=\tilde{v}\sigma$, $P''=P'\sigma$ and $P\equiv(\nu\,\tilde{s})(\bar{c}\langle\tilde{v}\rangle\,|\,P')$. Clearly, $P\xrightarrow{(\nu\,\tilde{s})\bar{c}\langle\tilde{v}\rangle}P'$, it implies $Q\xrightarrow{(\nu\,\tilde{s})\bar{c}\langle\tilde{v}\rangle}Q'$ and $P'\sim_{\kappa r}Q'$ since $P\sim_{\kappa r}Q$, this further implies there exists some Q' s.t. $Q\equiv(\nu\,\tilde{s})(\bar{c}\langle\tilde{v}\rangle\,|\,Q')$. Therefore, $Q\sigma\equiv(\nu\,\tilde{s})(\bar{m}\langle\tilde{u}\rangle\,|\,Q'\sigma)$ and $Q\sigma\xrightarrow{(\nu\,\tilde{s})\bar{m}\langle\tilde{u}\rangle}Q'\sigma$. This matches $(P'\sigma, Q'\sigma)\in\mathcal{R}$ as required.

$P\sigma\xrightarrow{{}^{+}m(\tilde{u})}P''$: since the substitution σ only affects to free output communication polars, this transition is only possible when $P{\downarrow}{}^{+}m(\tilde{v})$ where $\tilde{v}$ satisfies $\tilde{u}=\tilde{v}\sigma$. By the transition rules, it is only possible when P is in the form $P\equiv(\nu\tilde{n}_{\mathrm{P}})(\Lambda_{\mathrm{P}}\circ[!(\nu\,\kappa){}^{+}m(\tilde{x})L_{\mathrm{P}}.P_1\otimes G_{\mathrm{P}})|P_2)$ where $m\notin\tilde{n}_{\mathrm{P}}$, and $\Lambda_{\mathrm{P}}\xrightarrow{{}^{+}m\uparrow L_{\mathrm{P}}}\Lambda'_{\mathrm{P}}$ and $P\xrightarrow{{}^{+}m(\tilde{v})}P'$ where $P'\equiv(\nu\tilde{n}_{\mathrm{P}},\kappa)(\Lambda'_{\mathrm{P}}\circ[!(\nu\,\kappa){}^{+}m(\tilde{x})L_{\mathrm{P}}.P_1\otimes G_{\mathrm{P}}]|P_1\{\tilde{v}/\tilde{x}\}|P_2)$. It is easy to verify that $P''=P'\sigma$.

Since $P\sim_{\kappa r}Q$, we have that, $P\xrightarrow{{}^{+}m(\tilde{v})}P'$ implies

either $Q\xrightarrow{{}^{+}m(\tilde{v})}Q'$ and $P'\sim_{\kappa r}Q'$. It is only possible when $Q\equiv(\nu\tilde{n}_{\mathrm{Q}})(\Lambda_{\mathrm{Q}}\circ[!(\nu\,\kappa){}^{+}m(\tilde{y})L_{\mathrm{Q}}.Q_1\otimes G_{\mathrm{Q}}]\,|\,Q_2)$ where $m\notin\tilde{n}_{\mathrm{Q}}$, $\Lambda_{\mathrm{Q}}\xrightarrow{{}^{+}m\uparrow L_{\mathrm{Q}}}\Lambda'_{\mathrm{Q}}$, and $Q'\equiv(\nu\tilde{n}_{\mathrm{Q}},\kappa)(\Lambda'_{\mathrm{Q}}\circ[!(\nu\,\kappa){}^{+}m(\tilde{y})L_{\mathrm{Q}}.Q_1\otimes G_{\mathrm{Q}})|Q_1\{\tilde{v}/\tilde{y}\}|\,Q_2)$. It is easy to see, $Q\sigma\xrightarrow{{}^{+}m(\tilde{u})}Q'\sigma$ and $(P'\sigma, Q'\sigma)\in\mathcal{R}$;

or $Q\xrightarrow{\tau}Q'$ and $P'\sim_{\kappa r}[{}^{+}m\langle\tilde{v}\rangle]Q'$. Since in the κ-calculus, as in normal π-calculus, names in an internal action τ is always hidden, so $Q\sigma\xrightarrow{\tau}Q'\sigma$. Notice $([{}^{+}m\langle\tilde{v}\rangle]Q')\sigma\equiv[{}^{+}m\langle\tilde{u}\rangle]Q'\sigma$, we have $(P'\sigma,([{}^{+}m\langle\tilde{v}\rangle]Q')\sigma)\in\mathcal{R}$, or, $(P'\sigma,[{}^{+}m\langle\tilde{u}\rangle](Q'\sigma))\in\mathcal{R}$;

$P\sigma\xrightarrow{\hat{\kappa}}P''$: by transition rules, it is only possible when $P\sigma\equiv\hat{\kappa}\,|\,P''$, since output polar subsititution does not affect releasing keys, therefore there must exist some P' such that $P''=P'\sigma$ and $P\equiv\hat{\kappa}\,|\,P'$. This implies $P\xrightarrow{\hat{\kappa}}P'$. Since $P\sim_{\kappa r}Q$, we have $Q\xrightarrow{\hat{\kappa}}Q'$, and $P'\sim_{\kappa r}Q'$. Again by transition rules, it is only possible when $Q\equiv\hat{\kappa}\,|\,Q'$, that is $Q\sigma\equiv\hat{\kappa}\,|\,Q'\sigma$. Therefore $Q\sigma\xrightarrow{\hat{\kappa}}Q'\sigma$, and we have $(P'\sigma, Q'\sigma)\in\mathcal{R}$ as required.

$P\sigma\xrightarrow{\check{\kappa}}P''$: since the substitution σ only affects to free output communication polars, this transition is only possible when $P{\downarrow}\check{\kappa}$. By transition rules, the process P must be in the form $P\equiv\Lambda_{\mathrm{P}}\circ[G_{\mathrm{P}}]\,|\,P_1$, $J_{\mathrm{P}}=guard(G_{\mathrm{P}})$, $\Lambda_{\mathrm{P}}\xrightarrow{\check{\kappa}@J_{\mathrm{P}}}\Lambda'_{\mathrm{P}}$. Therefore, $P\xrightarrow{\check{\kappa}}P'$, where $P'=\Lambda'_{\mathrm{P}}\circ[G_{\mathrm{P}}]\,|\,P_1$. Since $P\sim_{\kappa r}Q$, this impies $Q\xrightarrow{\check{\kappa}}Q'$ and $P'\sim_{\kappa r}Q'$. By transition rules, process Q must be of the form $Q\equiv\Lambda_{\mathrm{Q}}\circ[G_{\mathrm{Q}}]\,|\,Q_1$, $\Lambda_{\mathrm{Q}}\xrightarrow{\check{\kappa}@J_{\mathrm{Q}}}\Lambda'_{\mathrm{Q}}$ and $Q'\equiv\Lambda'_{\mathrm{Q}}\circ[G_{\mathrm{Q}}]\,|\,Q_1$, where $J_{\mathrm{Q}}=guard(G_{\mathrm{Q}})$. It is easy to see, $P''=P'\sigma$ and $Q\sigma\xrightarrow{\check{\kappa}}Q'\sigma$. Therefore $(P'\sigma, Q'\sigma)\in\mathcal{R}$.

$P\sigma\xrightarrow{\tau}P''$: since the substitution σ only affects to free output communication polars, therefore this transition is only possible when $P\xrightarrow{\tau}P'$. By the transition rules, it must be in one of the following cases:

either there exist some process P_1 and complementary output-input action pair $\bar{\alpha}=(\nu\,\tilde{s})\bar{m}\langle\tilde{u}\rangle$ and ${}^{+}\alpha={}^{+}m(\tilde{u})$ such that, $P\equiv(\nu\,\tilde{s},m)(P_1|\bar{m}\langle\tilde{u}\rangle)$, $P_1\xrightarrow{{}^{+}m(\tilde{u})}P'_1$, and $P'\equiv(\nu\,\tilde{s},m)P'_1$. By previous results within this proof, it is easy to see that $P\sigma\xrightarrow{\tau}((\nu\,\tilde{s},m)P'_1)\sigma$, that is $P''\equiv P'\sigma$;

or there exist some process P_1 and complementary lock-unlock action pair $\hat{\kappa}$ and $\check{\kappa}$ such that, $P\equiv(\nu\,\kappa)(P_1|\hat{\kappa})$, $P_1\xrightarrow{\check{\kappa}}P'_1$, and $P'\equiv(\nu\,\kappa)P'_1$. By previous results within this proof, it is easy to see that $P\sigma\xrightarrow{\tau}((\nu\,\kappa)P'_1)\,\sigma$, that is $P''\equiv P'\sigma$.

Since $P\sim_{\kappa r}Q$, we also have that $P\xrightarrow{\tau}P'$ implies $Q\xrightarrow{\tau}Q'$ and $P'\sim_{\kappa r}Q'$. Then with the same way above, we always have $Q\sigma\xrightarrow{\tau}Q'\sigma$. Therefore $(P'\sigma, Q'\sigma)\in\mathcal{R}$.

Put all these cases together, we have that, $\mathcal{R}$ is a $\sim_{\kappa r}$ upto $\equiv_{\mathrm{P}}$. ■

Proofs for other properties of the responsive bisimulations also have the similar difference with those for standard bisimulations or conventional π–calculi, and we have to provide them all instantially.

Proposition 4−31: The responsive bisimulations are preserved by restriction. That is, to each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, whenever $P\mathcal{S}Q$ implies $(\nu\,\tilde{n})P\mathcal{S}(\nu\,\tilde{n})Q$ for all $\tilde{n}$.

Proof: These can be proven by showing $\mathcal{R}\overset{\mathrm{def}}{=}\{((\nu\,\tilde{n})P, (\nu\,\tilde{n})Q): P\mathcal{S}Q\}\cup\mathcal{S}$ is a $\mathcal{S}$ upto $\equiv$, where $\equiv$ be the structural congruences in Figure 2-3 and Figure 3-2. Again we only show that for strong κ-version $\mathcal{S}\subseteq\sim_{\kappa r}$, and all other cases can be proven similarly. Lets exam all the possible transitions that $(\nu\,\tilde{n})P$ may take:

$(\nu\,\tilde{n})P\xrightarrow{(\nu\tilde{s})\bar{m}\langle\tilde{u}\rangle}P''$: Giving the implicity renaming has removed all fresh name clash, let $\tilde{v}=\tilde{n}\cap\tilde{u}$ and $\tilde{n}_2=\tilde{n}-\tilde{v}$, then we have $(\nu\,\tilde{n})P\equiv(\nu\,\tilde{v})(\nu\,\tilde{n}_2)P$. From the structural congruence rules and transition rules, this transition is only possible when $m\notin\tilde{n}$, $\tilde{v}\subseteq\tilde{s}$ and there exists some P' such that $P\equiv(\nu\,\tilde{r})(P'|\bar{m}\langle\tilde{u}\rangle)$

where $\tilde{r}=\tilde{s}-\tilde{v}$. By rule tr-OUT, tr-RES and tr-PARL, we have $P\xrightarrow{(\nu\tilde{r})\bar{m}\langle\tilde{u}\rangle}P'$ and $P''\equiv(\nu\tilde{n}_2)P'$. This implies $Q\xrightarrow{(\nu\tilde{r})\bar{m}\langle\tilde{u}\rangle}Q'$ and $P'\sim_{\kappa r}Q'$ since $P\sim_{\kappa r}Q$, by rule tr-OUT and tr-RES we have $(\nu\tilde{n})Q\xrightarrow{(\nu\tilde{s})\bar{m}\langle\tilde{u}\rangle}(\nu\tilde{n}_2)Q'$, it matches $((\nu\tilde{n}_2)P',(\nu\tilde{n}_2)Q')\in\mathcal{R}$ as required.

$(\nu\tilde{n})P\xrightarrow{\dagger m(\tilde{u})}P''$: it is only possible when $m\notin\tilde{n}$ and $P\downarrow\dagger m(\tilde{u})$. Let $P\xrightarrow{\dagger m(\tilde{u})}P'$, by rule tr-RES, $(\nu\tilde{n})P\xrightarrow{\dagger m(\tilde{u})}(\nu\tilde{n})P'$, that is, $P''\equiv(\nu\tilde{n})P'$. From $P\sim_{\kappa r}Q$, the transition $P\xrightarrow{\dagger m(\tilde{u})}P'$ implies

either $Q\xrightarrow{\dagger m(\tilde{u})}Q'$ and $P'\sim_{\kappa r}Q'$. By rule tr-RES, $(\nu\tilde{n})Q\xrightarrow{\dagger m(\tilde{u})}(\nu\tilde{n})Q'$, therefore $((\nu\tilde{n})P',(\nu\tilde{n})Q')\in\mathcal{R}$;

or $Q\xrightarrow{\tau}Q'$ and $P'\sim_{\kappa r}[\dagger m\langle\tilde{u}\rangle]Q'$. Then $(\nu\tilde{n})Q\xrightarrow{\tau}(\nu\tilde{n})Q'$ by rule tr-RES, and it also match $((\nu\tilde{n})P',(\nu\tilde{n})([\dagger m\langle\tilde{u}\rangle]Q'))\in\mathcal{R}$ as required.

$(\nu\tilde{n})P\xrightarrow{\hat{\kappa}}P''$: it is only possible when $\kappa\notin\tilde{n}$ and $P\xrightarrow{\hat{\kappa}}P'$, that implies $Q\xrightarrow{\hat{\kappa}}Q'$ and $P'\sim_{\kappa r}Q'$, since $P\sim_{\kappa r}Q$. From rule tr-RES, $P''\equiv(\nu\tilde{n})P'$ and $(\nu\tilde{n})Q\xrightarrow{\hat{\kappa}}(\nu\tilde{n})Q'$, and therefore $((\nu\tilde{n})P',(\nu\tilde{n})Q')\in\mathcal{R}$ as required.

$(\nu\tilde{n})P\xrightarrow{\check{\kappa}}P''$: by rule tr-RES, tr-RELS and str-SCP3, it is only possible when $\kappa\notin\tilde{n}$ and $P\xrightarrow{\check{\kappa}}P'$, that implies $Q\xrightarrow{\check{\kappa}}Q'$ and $P'\sim_{\kappa r}Q'$, since $P\sim_{\kappa r}Q$. From rule tr-RES we have $P''\equiv(\nu\tilde{n})P'$ and $(\nu\tilde{n})Q\xrightarrow{\check{\kappa}}(\nu\tilde{n})Q'$, and therefore $((\nu\tilde{n})P',(\nu\tilde{n})Q')\in\mathcal{R}$ as required.

$(\nu\tilde{n})P\xrightarrow{\tau}P''$: it is only possible when $P\xrightarrow{\tau}P'$, then we have $Q\xrightarrow{\tau}Q'$ and $P'\sim_{\kappa r}Q'$, since $P\sim_{\kappa r}Q$. From rule tr-RES we have $P''\equiv(\nu\tilde{n})P'$ and $(\nu\tilde{n})Q\xrightarrow{\tau}(\nu\tilde{n})Q'$, and therefore $((\nu\tilde{n})P',(\nu\tilde{n})Q')\in\mathcal{R}$ as required.

Put them together, by the definition, $\mathcal{R}$ is a $\sim_{\kappa r}$ upto $\equiv$. ■

The following proposition is equivelant to say, in the term of ordinary π–calculi, the responsive bisimulations are preserved by input prefix, replication, choice and, outside the π–calculi scope, lock, for autonomous processes.

Proposition 4−32: The responsive bisimulations are preserved by *GEC* choice for autonomous processes. That is, to each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, if P_1 and P_2 are autonomous processes, then $P_1\mathcal{S}P_2$ implies $\mathcal{D}[P_1]\mathcal{S}\mathcal{D}[P_2]$ for all process context $\mathcal{D}[.]$ of the form $\mathcal{D}[.]\stackrel{\text{def}}{=}\Lambda\circ[\,!(\nu\kappa)\dagger m(\tilde{x})L.[.]\otimes G]$.

Proof: In order to prove the proposition, we extend it to a more generic form:

Let R be an arbitrary process, R_1 and R_2 be arbitrary safe processes (this means $fin(R_1|R_2)\cap(fin(R)\cup fin(G)\cup\{m\})=\varnothing$) satisfying $R_1\mathcal{S}R_2$, and $\tilde{\kappa}_1$ be set of keys, if we can prove that, "$P_1\mathcal{S}P_2$ implies $(\nu\tilde{\kappa}_1)(\mathcal{D}[P_1]|R|R_1)\mathcal{S}(\nu\tilde{\kappa}_1)(\mathcal{D}[P_2]|R|R_2)$ for all such $\mathcal{D}[.]$, R, R_1, R_2, and $\tilde{\kappa}_1$", then this proposition can be concluded by letting $R=\mathbf{0}$, $R_1=\mathbf{0}$, $R_2=\mathbf{0}$ and $\tilde{\kappa}_1=\varnothing$.

Let $safe(P,m,G,R)$ be a function giving the truth value of $fin(P)\cap(fin(R)\cup fin(G)\cup\{m\})=\varnothing$, let $\mathcal{P}_a$ be the set of all autonomous processe, then we show that

$$\mathcal{R}\stackrel{\text{def}}{=}\{\,((\nu\tilde{\kappa}_1)(\Lambda\circ[\,!(\nu\kappa)\dagger m(\tilde{x})L.P_1\otimes G]|R|R_1),\ (\nu\tilde{\kappa}_1)(\Lambda\circ!(\nu\kappa)\dagger m(\tilde{x})L.P_2\otimes G]|R|R_2)):\ (P_1,P_2\in\mathcal{P}_a)\wedge(P_1\mathcal{S}P_2)\wedge safe(R_1|R_2,m,G,R)\wedge(R_1\mathcal{S}R_2)\,\}$$

is a $\mathcal{S}$. To save our writing, we define the *GEC* context $\mathcal{G}[.]\stackrel{\text{def}}{=}!(\nu\kappa)\dagger m(\tilde{x})L.[.]\otimes G$, that is, $\mathcal{D}[.]\equiv\Lambda\circ[\mathcal{G}[.]]$, and write

$$Q_1\stackrel{\text{def}}{=}(\nu\tilde{\kappa}_1)(\Lambda\circ[\mathcal{G}[P_1]]|R|R_1)\equiv(\nu\tilde{\kappa}_1)(\mathcal{D}[P_1]|R|R_1)\quad\text{and}\quad Q_2\stackrel{\text{def}}{=}(\nu\tilde{\kappa}_1)(\Lambda\circ[\mathcal{G}[P_2]]|R|R_2)\equiv(\nu\tilde{\kappa}_1)(\mathcal{D}[P_2]|R|R_2),$$

and then show "whenever $P_1\mathcal{S}P_2$ implies $Q_1\mathcal{S}Q_2$" by induction over all possible actions. Here we only show that for $\mathcal{S}\subseteq\sim_{\kappa r}$, all others can be proven similarly.

Whenever $Q_1\xrightarrow{\alpha}Q_1'$ for some action α, then it must satisfy $fn(\alpha)\cap\tilde{\kappa}_1=\varnothing$, and is only possible in one of the following four cases:

1), $R\xrightarrow{\alpha}R'$, then $Q_1'\stackrel{\text{def}}{=}(\nu\tilde{\kappa}_1)(\mathcal{D}[P_1]|R'|R_1)$ and $Q_2\xrightarrow{\alpha}Q_2'$, $Q_2'\stackrel{\text{def}}{=}(\nu\tilde{\kappa}_1)(\mathcal{D}[P_2]|R'|R_2)$, and $(Q_1',Q_2')\in\mathcal{R}$.

2), $R_1\xrightarrow{\alpha}R_1'$, then $Q_1'\stackrel{\text{def}}{=}(\nu\tilde{\kappa}_1)(\mathcal{D}[P_1]|R|R_1')$. If α is not a communication input action, then $R_2\xrightarrow{\alpha}R_2'$ and $R_1'\mathcal{S}R_2'$ since $R_1\mathcal{S}R_2$. Therefore $Q_2\xrightarrow{\alpha}Q_2'$ where $Q_2'\stackrel{\text{def}}{=}(\nu\tilde{\kappa}_1)(\mathcal{D}[P_1]|R|R_2')$, and $(Q_1',Q_2')\in\mathcal{R}$.

3), $\Lambda\circ[\mathcal{G}[P_1]]\downarrow\alpha$, lets exam all the possible action α:

$\alpha=\check{\kappa}'$, it must be $\kappa'\notin\tilde{\kappa}_1$ and $\Lambda\xrightarrow{\check{\kappa}'@J}\Lambda'$, where $J\subseteq\{m\}\cup guard(G)$, by rule tr-RELS we have $Q_1'\stackrel{\text{def}}{=}(\nu\tilde{\kappa}_1)(\Lambda'\circ[\mathcal{G}[P_1]]|R|R_1)$ and $Q_2\xrightarrow{\check{\kappa}'}Q_2'$ where $Q_2'\stackrel{\text{def}}{=}(\nu\tilde{\kappa}_1)(\Lambda'\circ[\mathcal{G}[P_2]]|R|R_2)$, therefore $(Q_1',Q_2')\in\mathcal{R}$;

$\alpha=\dagger m(\tilde{u})$, it must be $\Lambda \xrightarrow{\dagger m \Uparrow L} \Lambda'$ and $Q_1' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1,\kappa)(\Lambda'\circ[\mathcal{G}[P_1]] \,|\, R \,|\, R_1 \,|\, P_1\{\tilde{u}/\tilde{x}\})$ and $\Lambda' \equiv addl(L,\Lambda)$, then Q_2 can also commit the same action such that $Q_2 \xrightarrow{\alpha} Q_2'$ and $Q_2' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1,\kappa)(\Lambda'\circ[\mathcal{G}[P_2]] \,|\, R \,|\, R_2 \,|\, P_2\{\tilde{u}/\tilde{v}\})$. Since P_1 and P_2 are autonomous, and therefore $P_1\{\tilde{u}/\tilde{x}\}$ and $P_2\{\tilde{u}/\tilde{x}\}$ are, this further implies that $R_1|P_1\{\tilde{u}/\tilde{x}\}$ and $R_2|P_2\{\tilde{u}/\tilde{x}\}$ are safe, that is, $safe((R_1|P_1\{\tilde{u}/\tilde{x}\})|(R_2|P_2\{\tilde{u}/\tilde{x}\}),m,G,R)$ is true. Since $P_1 \mathcal{S} P_2$ and $R_1 \mathcal{S} R_2$, by Proposition 4-30 and Proposition 4-29, we have $(R_1|P_1\{\tilde{u}/\tilde{x}\}) \mathcal{S} (R_2|P_2\{\tilde{u}/\tilde{x}\})$, therefore $(Q_1', Q_2') \in \mathcal{R}$;

$\alpha=\dagger m_1(\tilde{u})$ where $m_1 \neq m$, it must be $\Lambda\circ[G] \xrightarrow{\alpha} (\nu\, \kappa'')(\Lambda'\circ[G] \,|\, P)$, then $Q_1' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1,\kappa'')(\Lambda'\circ[\mathcal{G}[P_1]] \,|\, (P|R) \,|\, R_1)$, and we can have $Q_2 \xrightarrow{\alpha} Q_2'$ and $Q_2' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1,\kappa'')(\Lambda'\circ[\mathcal{G}[P_2]] \,|\, (P|R) \,|\, R_2)$. Therefore, $(Q_1', Q_2') \in \mathcal{R}$.

4), α is a τ action caused by a synchronisation action between $\mathcal{D}[P_1]$, R, R_1, following the rule tr-SYNC2. This in turn can be only possible in the following sub-cases which satisfy $\kappa' \in \tilde{\kappa}_1$:

$\Lambda\circ[\mathcal{G}[P_1]]\downarrow\check{\kappa}'$, it must be $\Lambda \xrightarrow{\check{\kappa}' @ J} \Lambda'$ where $J \subseteq \{m\} \cup guard(G)$, and
either $R \xrightarrow{\hat{\kappa}'} R'$, then $Q_1' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1)(\Lambda'\circ[\mathcal{G}[P_1]] \,|\, R' \,|\, R_1)$ and $Q_2 \xrightarrow{\tau} Q_2'$ where $Q_2' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1)(\Lambda'\circ[\mathcal{G}[P_2]] \,|\, R' \,|\, R_2)$, therefore $(Q_1', Q_2') \in \mathcal{R}$;
or $R_1 \xrightarrow{\hat{\kappa}'} R_1'$, then $Q_1' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1)(\Lambda'\circ[\mathcal{G}[P_1]] \,|\, R \,|\, R_1')$, and we have $R_2 \xrightarrow{\hat{\kappa}'} R_2'$ and $R_1' \mathcal{S} R_2'$ since $R_1 \mathcal{S} R_2$. Therefore $Q_2 \xrightarrow{\tau} Q_2'$ where $Q_2' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1)(\Lambda'\circ[\mathcal{G}[P_2]] \,|\, R \,|\, R_2')$, and we have $(Q_1', Q_2') \in \mathcal{R}$;

$R \xrightarrow{\check{\kappa}'} R'$, $R_1 \xrightarrow{\hat{\kappa}'} R_1'$, and we have $R_2 \xrightarrow{\hat{\kappa}'} R_2'$ and $R_1' \mathcal{S} R_2'$ since $R_1 \mathcal{S} R_2$. That is, $Q_1' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1)(\mathcal{D}[P_1] \,|\, R' \,|\, R_1')$ and $Q_2 \xrightarrow{\tau} Q_2'$ where $Q_2' \stackrel{\text{def}}{=} (\nu\, \tilde{\kappa}_1)(\mathcal{D}[P_1] \,|\, R' \,|\, R_2')$, therefore $(Q_1', Q_2') \in \mathcal{R}$;

Put them together, we have $\mathcal{R} \subseteq \mathcal{S}$ by the definition of $\sim_{\kappa r}$. ∎

For generic safe processes the situation becomes complicated, because the safe condition can be broken when an safe process is duplicated by replication, and needs closer studies in the future works. When replications are erased by lock, then the preservation will be certain:

Lemma 4−33: For each κ-version or non-κ-version responsive bisimulation $\mathcal{S}$, if P_1 and P_2 are safe processes, then $P_1 \mathcal{S} P_2$ implies $\mathcal{D}[P_1] \mathcal{S} \mathcal{D}[P_2]$ for all process context of the form $\mathcal{D}[.] \stackrel{\text{def}}{=} \Lambda\circ[\dagger m(\tilde{x})(\nu)@J.[.] \otimes G]$ where $m \in J$.
Proof: Similar to that of Proposition 4-32 but simpler, since only one copy of P_1 and P_2 involved. ∎

Proposition 4−34: For autonomous processes, the responsive bisimulations are congruences. That is, for each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, if P_1 and P_2 are autonomous processes, then $P_1 \mathcal{S} P_2$ implies $\mathcal{C}[P_1] \mathcal{S} \mathcal{C}[P_2]$ for all process context $\mathcal{C}[.]$.
Proof: Immediately concluded by put Proposition 4-29, Proposition 4-30 and Proposition 4-32 together. ∎

5 Discussion

5.1 Coincidence between some variations of responsive bisimulation

The early, late and open concepts used for bismulations in standard π-calculus, may apply to responsive bisimulation.

Definition 5−36: Each of the following variations of responsive bisimulations is a (strong) oτ-bisimulation $\mathcal{S}$:

The (strong) **early responsive bisimulation** is a (strong) oτ-bisimulation $\mathcal{S}$ if whenever $P \mathcal{S} Q$ then $P \xrightarrow{\dagger m(\tilde{u})} P'$ implies $\forall \tilde{y}\ \exists Q'$ s.t. either $Q \xrightarrow{\dagger m(\tilde{u})} Q'$ and $P'\{\tilde{y}/\tilde{u}\} \mathcal{S} Q'\{\tilde{y}/\tilde{u}\}$, or $Q \xrightarrow{\tau} Q'$ and $P'\{\tilde{y}/\tilde{u}\} \mathcal{S} ([\dagger m\langle\tilde{u}\rangle]Q')\{\tilde{y}/\tilde{u}\}$;

The (strong) **late responsive bisimulation** is a (strong) oτ-bisimulation $\mathcal{S}$ if whenever $P \mathcal{S} Q$ then $P \xrightarrow{\dagger m(\tilde{u})} P'$ implies $\exists Q'$ s.t. either $Q \xrightarrow{\dagger m(\tilde{u})} Q'$ and $\forall \tilde{y}, P'\{\tilde{y}/\tilde{u}\} \mathcal{S} Q'\{\tilde{y}/\tilde{u}\}$, or $Q \xrightarrow{\tau} Q'$ and $\forall \tilde{y}, P'\{\tilde{y}/\tilde{u}\} \mathcal{S} ([\dagger m\langle\tilde{u}\rangle]Q')\{\tilde{y}/\tilde{u}\}$;

The (strong) **open responsive bisimulation** is a symmetric relation $\mathcal{S}$ on processes if whenever $P \mathcal{S} Q$ then for any output communication polar substitution $\sigma=\{\tilde{y}/\tilde{x}\}$, we have
$P\sigma \xrightarrow{\alpha} P'$ implies $\exists Q'$ s.t. $Q\sigma \xrightarrow{\alpha} Q'$ and $P' \mathcal{S} Q'$, where either $\alpha=(\nu\, \tilde{v})\bar{m}\langle\tilde{u}\rangle$ and $\tilde{v} \cap fn(Q)=\emptyset$, or $\alpha=\tau$;

$P\xrightarrow{{}^{\dagger}m(\tilde{u})}P'$ implies $\exists Q'$ s.t. either $Q\sigma\xrightarrow{{}^{\dagger}m(\tilde{u})}Q'$ and $P'\mathcal{S}Q'$ or $Q\sigma\xrightarrow{\tau}Q'$ and $P'\mathcal{S}[{}^{\dagger}m\langle\tilde{u}\rangle]Q'$.

such that whenever $P\mathcal{S}Q$ then $P\xrightarrow{\alpha}P'$ implies $Q\xrightarrow{\alpha}Q'$ and $P'\mathcal{S}Q'$ for all action α in the form of either $\alpha=(\nu\,\tilde{v})\overline{m}\langle\tilde{u}\rangle$ or $\alpha=\tau$, and $bn(\alpha)\cap fn(Q)=\emptyset$.

For early and late responsive bisimulations, the κ-versions are defined by replace οτ-bisimulation with κοτ-bisimulation in the above definitions. For open responsive bisimulations, the κ-versions are defined by including ${}^{+}\mathcal{K}\cup{}^{-}\mathcal{K}$ into the range of α.

However, as the κ-calculus is an asynchronised process algebra, these variations are not necessary for it, since they coincide with the standard version of responsive bisimulation.

Lemma 5−37: The early, late and open responsive bisimulations all coincide with the standard version of responsive bisimulation.

Proof: The proof is trivial. For the non-κ-versions, let $\mathcal{S}_r$ be strong (or weak) responsive bisimulation, $\mathcal{S}$ be any of strong (or weak, respectively) early, late and open responsive bisimulations, then we got

$P\mathcal{S}Q$ implies $P\mathcal{S}_rQ$, by letting $\tilde{y}=\tilde{u}$; and $P\mathcal{S}_rQ$ implies $P\mathcal{S}Q$, by applying Proposition 4-30.

Similarly we can prove it for the κ-versions. ■

5.2 Relation with some conventional bisimulations

Most familiar bisimulation relations which are widely used in convensional π-calculus can be also defined in the κ-calculus, with the similar style as we did for the polar π-calculus ([Zhang01A]).

Definition 5−38: The (strong) **ground bisimulation** is a (strong) οτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then $P\xrightarrow{{}^{\dagger}m(\tilde{u})}P'$ implies either $Q\xrightarrow{{}^{\dagger}m(\tilde{u})}Q'$.

The (strong) **early bisimulation** is a (strong) οτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then $P\xrightarrow{{}^{\dagger}m(\tilde{u})}P'$ implies $\forall\tilde{y}\ \exists Q'$ s.t. $Q\xrightarrow{{}^{\dagger}m(\tilde{u})}Q'$ and $P'\{\tilde{y}/\tilde{u}\}\mathcal{S}\,Q'\{\tilde{y}/\tilde{u}\}$;

The (strong) **late bisimulation** is a (strong) οτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then $P\xrightarrow{{}^{\dagger}m(\tilde{u})}P'$ implies $\exists Q'$ s.t. $Q\xrightarrow{{}^{\dagger}m(\tilde{u})}Q'$ and $\forall\tilde{y}$ $(P'\{\tilde{y}/\tilde{u}\}\mathcal{S}\,Q'\{\tilde{y}/\tilde{u}\})$;

The (strong) **open bisimulation** is a (strong) οτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then for any output name substitution $\sigma=\{\tilde{y}/\tilde{x}\}$, $P\sigma\xrightarrow{\alpha}P'$ implies $\exists Q'$ s.t. $Q\sigma\xrightarrow{\alpha}Q'$ and $P'\mathcal{S}Q'$;

For each of them the weak version is obtained by replacing transitions with weak transitions everywhere, and the κ-version is defined by replace οτ-bisimulation with κοτ-bisimulation in the above definitions. We denote $\sim_{\kappa g}$ ($\approx_{\kappa g}$) and $\sim_g$ ($\approx_g$) be the largest strong (weak) κ-version and non-κ-version ground bisimulation respectively.

Lemma 5−39: The ground bisimulation, early bisimulation, late bisimulation and open bisimulation are all coincided in the κ-calculus.

Proof: First, the ground bisimulations are preserved by output polarity name substitution, this can be proven in a way similar to that for Proposition 4-30, except no need to check the cases involving localisation, then the lemma is followed. ■

Corollary 5−40: The ground bisimulations are responsive bisimulations, that is, $\sim_g\subseteq\sim_r$ and $\approx_g\subseteq\approx_r$.

Proof: Directly concluded from the comparison of their definitions. ■

The asynchronous bisimulation of [Amadio96], which emphasises the possible delay of message delivery (output) and allows the sent message moving around within a communication channel without real information exchange, can also be described in the κ-calculus:

Definition 5−41: The (strong) **asynchronous bisimulation** is a (strong) οτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then $P \xrightarrow{\dagger m(\tilde{u})} P'$ implies either $Q \xrightarrow{\dagger m(\tilde{u})} Q'$ and $P' \sim_{\mathrm{a}} Q'$, or $Q \xrightarrow{\tau} Q'$, and $P'\mathcal{S}(\bar{m}\langle\tilde{u}\rangle \mid Q')$.

Again, the weak asynchronous bisimulation is obtained by replacing transitions with weak transitions everywhere, and the κ-version is defined by replace οτ-bisimulation with κοτ-bisimulation. We denote $\sim_{\kappa\mathrm{a}}$ ($\approx_{\kappa\mathrm{a}}$) and $\sim_{\mathrm{a}}$ ($\approx_{\mathrm{a}}$) be the largest strong (weak) κ-version and non-κ-version asynchronous bisimulation respectively.

As pointed out in [Zhang01A], both the responsive bisimulation and asynchronous bisimulation describe asynchronous communication by allowing message delay. They are overlapped, but none of them contains another, as shown in the Figure 5-2. The asynchronous bisimulation is not interested in because the following reasons:

1. We are interested in the delay of input rather then that of output;
2. To capture the delay of output, the asynchronous bisimulation allows competition on grabbing messages from the same input port, which can disturb the detection of responsive behaviours;
3. Combining both output delay and input delay will make the theory unnecessary complicated.

In contrary, the responsive bisimulation concentrates on the delay of input. In the view of object-oriented programming, the delay in the delivery is not visible for either sender or receiver, and is also out of their control. The delay of input, however, is controllable for the receiver, and, as pointed out by [McHale94] and [Zhang98B], the existence of the interval between the event of a message arriving an object and the event of the message processing starts, provides a synchronisation control point for compositional concurrent object. In other words, the responsive bisimulation is quite natural to compositional objects.

5.3 Relation between κ-version and non-κ-version bisimulations

For each bisimulations we have studied in the κ-calculus, its κ-version is a subset of its non-κ-version, according to their definitions. Generally, when modelling objects, the scope of a lock key κ should not cross object boundary, and therefore the $\hat{\kappa}$ and $\check{\kappa}$ actions that an object can take are internal to that object and can not be detected from outside. The locking and unlocking signals represent a special kind of communication, or, co-ordination, between components within an object, and responsible for whether, why, when and how messages be delayed from inputting to the object. Taking the internal view of objects, the κ-version bisimulations guarantee the similarity of co-ordination mechanism, and therefore the replaceability of object components. In contrary, non-κ-version bisimulations confirm the similarity of overall behaviour between objects, without knowing the details of the co-ordination mechanisms. When measurement of the behaviour of objects or object groups is restricted to external view, then the κ-version and non-κ-version of a bisimulation will coincide.

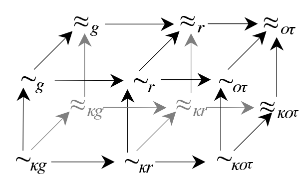


Figure 5-1

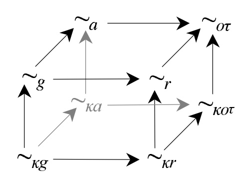


Figure 5-2

The Figure 5-1 and Figure 5-2 summarise some bisimulation relations discussed so far, from the strongest one, $\sim_{\kappa g}$, to the weakest, $\approx_{o\tau}$, where each arrow respresnets a "⊆", or "is a subset of", relation.

5.4 The relation between the responsive bisimulation in the polar π-calculus and in the κ-calculus

The concept of responsive bisimulation was simpler when described in the polar π-calculus, than that in the κ-calculus. That is because: 1) only has the simplest choice cases to handle; 2) the κ-version of responsive bisimulation is merged into the non-κ-version, since the locking signals either become ordinary communication, or are hidden by the synatx of choice; 3) smaller syntax.

As we have already pointed at the beginning, the polar π-calculus, which is a sub-calculus of the κ-calculus, is not an idea tool for modelling compositional objects. From object modelling point of view, the responsive bisimulation in the polar π-calculus actually overlaps with both the κ-version and non-κ-version of that in the κ-calculus. However, the definition of responsive bisimulation in the polar π-calculus has no difference with the non-κ-version responsive bisimulation in the κ-calculus, and therefore providers a simplified platform to describe the properties of the latter.

6 Application

In the client's eyes, the objects O_1 and O_2 of the Figure 1-1 are behaviourally the same. With the responsive bisimulation, we express this as $O_1 \approx_{\kappa r} O_2$.

By adopting the higher order G-terms, the behaviour of concurrent objects modelled in the κ-calculus can be separated even further. The behaviour of a generic form of object may be modelled as

$$O_G \overset{\text{def}}{=} (\tilde{m})\lfloor\,\rfloor\circ[\mathbb{G}«\tilde{M}»\langle\tilde{\overset{\dagger}{m}}\rangle], \text{ where } \mathbb{G} \overset{\text{def}}{=} «\tilde{\eta}»\,(\tilde{\overset{\dagger}{n}})\bigotimes_{i\in I}!(\nu\,\kappa)\,\overset{\dagger}{n}_i(\tilde{\bar{v}})\,\check{\kappa}@J_i.\eta_i\langle\tilde{\bar{v}},\hat{\kappa}\rangle$$

$\mathbb{G}$ expresses purely for the exclusion, $\tilde{m}$ are the method names and $\tilde{M}$ represent the behaviours of methods' body. We may consider the context within the ⌈ ⌉ brackets represents the static behaviour (interface/definition) of objects, since it never change with reduction. For a compositional object, the model becomes

$$O_C \overset{\text{def}}{=} (\tilde{m})(\nu\,\tilde{n})(C\langle\tilde{\overset{\dagger}{m}},\tilde{n}\rangle \mid F\langle\tilde{n}\rangle),$$

$$\text{where } F \overset{\text{def}}{=} (\tilde{n})\lfloor\,\rfloor\circ[\mathbb{E}«\tilde{M}»\langle\tilde{\overset{\dagger}{n}}\rangle], \qquad \mathbb{E} \overset{\text{def}}{=} «\tilde{\eta}»(\tilde{\overset{\dagger}{n}})\bigotimes_{i\in I}!\overset{\dagger}{n}_i(\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{v}})(\nu)@\emptyset.\eta_i\langle\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{v}}\rangle$$

$$C \overset{\text{def}}{=} (\tilde{\overset{\dagger}{m}},\tilde{n})\lfloor\,\rfloor\circ[\mathbb{G}«\{S_i\langle\bar{n}_i\rangle_{i\in I}\}»\langle\tilde{\overset{\dagger}{m}}\rangle], \qquad \mathbb{G} \overset{\text{def}}{=} «\tilde{\eta}»(\tilde{\overset{\dagger}{m}})\bigotimes_{i\in I}!(\nu\,\kappa)\overset{\dagger}{m}_i(\bar{s}_m,\bar{s}_f,\bar{r}_m,\bar{t}_m,\tilde{\bar{v}})\,\check{\kappa}@J_i.\eta_i\langle\bar{s}_m,\bar{s}_f,\bar{r}_m,\bar{t}_m,\tilde{\bar{v}},\hat{\kappa}\rangle$$

F presenting the functionality of O_C in the form of an object without any constraint, the higher order G-term $\mathbb{E}$ gives no exclusion constraint, C presents the concurrency constraints and consists the exclusion control $\mathbb{G}$ and the schedulers $\tilde{S}$. An example of scheduler can be $S_i \overset{\text{def}}{=} (\bar{n}_i,\bar{s}_m,\bar{s}_f,r_m,t_m,\tilde{\bar{v}},\hat{\kappa})\,(\nu\ s_n,r_n,t_n)\,(\bar{s}_m \mid \bar{n}_i\langle\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{v}}\rangle \mid \overset{\dagger}{s}_n.\,\overset{\dagger}{r}_n(\tilde{u}).\,(\bar{r}_m\langle\tilde{u}\rangle \mid \overset{\dagger}{t}_n.(\bar{t}_m \mid \hat{\kappa}))))$. In this model, the scope of the lock/unlock signal κ is never beyond the controller C. An extension to concurrent object-oriented programming languages based on this model is presented in [Zhang01D]. Now we can use $O_C \approx_r O_G$ to express the behaviour equivalence between the compositional object O_C and the non-compositional object O_G. Here we are able to use $\approx_r$ rather than $\approx_{\kappa r}$ because in these models the locks are private to objects.

To investigate the properties of object composition further, we introduce some more terminologies and symbol.

Definition 6−42: The safe process P is an *object component process* with *source set* $\tilde{\overset{\dagger}{m}}$, where $\{\tilde{\overset{\dagger}{m}}\}=fin(P)$, if $P\downarrow\alpha$ for all $a\in\tilde{\overset{\dagger}{m}}$.

The object component process C with source set $\tilde{\overset{\dagger}{n}}$ is a *control process* with *socket set* $\tilde{\overset{\dagger}{n}}$ and *plug set* $\tilde{\bar{m}}$, if $\{\tilde{\bar{m}}\}\subseteq fn(C)$, $\{\tilde{m}\}\cap\{\tilde{n}\}=\emptyset$, and for each i where $\overset{\dagger}{n}_i\in\tilde{\overset{\dagger}{n}}$ and $\bar{m}_i\in\tilde{\bar{m}}$, there exist processes C', C'' and action sequence ℓ which satisfies $\{\tilde{\overset{\dagger}{n}},\tilde{\bar{m}}\}\cap fn(\ell)=\emptyset$, such that $C\langle\tilde{\overset{\dagger}{n}},\tilde{\bar{m}}\rangle \xrightarrow{\overset{\dagger}{n}_i(\tilde{u})} C'$, $C' \xrightarrow{\ell} C''$ and $C''\downarrow\bar{m}_i\langle\tilde{u}\rangle$.

We define the generic empty control process as $E \overset{\text{def}}{=} (\tilde{n},\tilde{m})\lfloor\,\rfloor\circ[\mathbb{E}«\{S_{Ei}\langle\bar{m}_i\rangle\}_{i\in I}»\langle\tilde{\overset{\dagger}{n}}\rangle]$ where

empty exclusion : $\mathbb{E} \overset{\text{def}}{=} «\tilde{\eta}»(\tilde{n})\bigotimes_{i\in I}!(\nu\,\kappa)\,\overset{\dagger}{n}_i(s,s_f,t,r,\tilde{\bar{v}})\,(\nu)@\emptyset.\,\eta_i\langle s,s_f,t,r,\tilde{\bar{v}}\rangle$

empty scheduler : $S_{Ei} \overset{\text{def}}{=} (\bar{m}_i,s,s_f,t,r,\tilde{v}).(\nu\ s',t',r')\bar{m}_i\langle\bar{s}',\bar{s}_f,\bar{t}',\bar{r}',\tilde{v}\rangle.\overset{\dagger}{s}'.\bar{s} \mid \overset{\dagger}{r}'(\tilde{u}).\bar{r}\langle\tilde{u}\rangle \mid \overset{\dagger}{t}'.\bar{t}\ )$

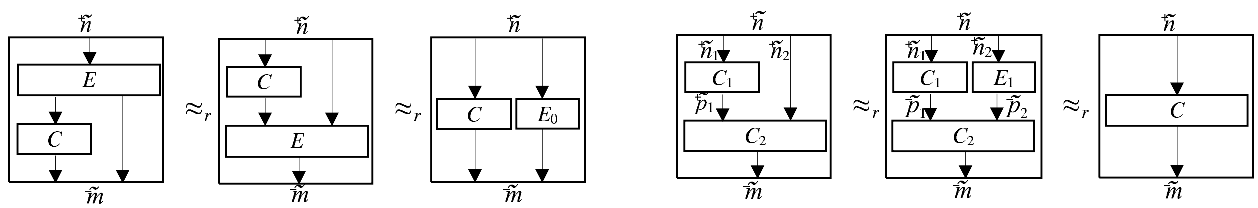
Given a control process D with socket set $\tilde{\overset{\dagger}{x}}$ and plug set $\tilde{\bar{y}}$ and an object component process Q with source set $\tilde{\overset{\dagger}{z}}$, let $C \overset{\text{def}}{=} (\tilde{\overset{\dagger}{x}},\tilde{\bar{y}})D$ and $P \overset{\text{def}}{=} (\tilde{\overset{\dagger}{z}})D$, we use the abbreviation $C \succ P$ to denote $(\tilde{m})(\nu\,\tilde{n})(C\langle\tilde{\overset{\dagger}{m}},\tilde{n}\rangle \mid P\langle\tilde{n}\rangle)$ for all $\tilde{m}$ and $\tilde{n}$, where $\{\tilde{m},\tilde{n}\}\cap fn(D)=\emptyset$ and $\{\tilde{m},\tilde{n}\}\cap fn(P)=\emptyset$.

One of the deserved properties of the composition is identity law. With ground bisimulation, [Zhang98A] has proven the identity law for right side $C \succ E \approx_g C$, but the left side law ($E \succ C \approx_g C$) is not generally true. With the responsive bisimulation, however, identity law is held for both sides: $E \succ C \approx_r C \succ E \approx_r C$, proved by [Zhang01C]. This property not only gives the mathematical elegance, or reflects the fact that adding an empty behaviour to a server object will make no difference in the clients' eyes, but more importantly, it means that we can always add new constrains to the existing control with relatively simple composition, without introducing unexpected side effect in behaviour. For example, assume the control process C_1 describes and only describes the exclusion between $\overset{\dagger}{m}_1$ and $\overset{\dagger}{m}_2$, and the control process C_2 describes and only describes the exclusion between $\overset{\dagger}{m}_2$ and $\overset{\dagger}{m}_3$, then $C_1 \succ C_2$ will provider both exclusion between $\overset{\dagger}{m}_1$ and $\overset{\dagger}{m}_2$, and that between $\overset{\dagger}{m}_2$ and $\overset{\dagger}{m}_3$, but no other exclusion will be accidentally added or removed. In a more formal and generic form, let $\mathbb{G}' \overset{\text{def}}{=} «\tilde{\eta}»(\tilde{\overset{\dagger}{x}})\bigotimes_{i\in I}!(\nu\,\kappa'_i)\overset{\dagger}{x}_i(\tilde{\bar{v}})\,\check{\kappa}'_i@J'_i.\eta_i\langle\tilde{\bar{v}},\hat{\kappa}'_i\rangle$, $\mathbb{G}'' \overset{\text{def}}{=} «\tilde{\eta}»(\tilde{\overset{\dagger}{x}})\bigotimes_{i\in I}!(\nu\,\kappa''_i)\overset{\dagger}{x}_i(\tilde{\bar{v}})\,\check{\kappa}''_i@J''_i.\eta_i\langle\tilde{\bar{v}},\hat{\kappa}''_i\rangle$ and $S'_i=S''_i \overset{\text{def}}{=} (\bar{x},\bar{s},\bar{s}_f,r,t,\tilde{u},\hat{\kappa})\,(\nu\ s_1,r_1,t_1)\,(\bar{x}\langle\bar{s}_1,\bar{s}_f,\bar{r}_1,\bar{t}_1,\tilde{u}\rangle \mid \overset{\dagger}{s}_1.\,(\bar{s} \mid (\overset{\dagger}{r}_1(\tilde{w}).\,(\bar{r}\langle\tilde{w}\rangle \mid \hat{\kappa} \mid (\overset{\dagger}{t}_1.\bar{t})))))$, then, proven by [Zhang01C],

$$(\nu\ \tilde{p})\,(\lfloor\,\rfloor\circ[\mathbb{G}'«\{S'_i\langle\bar{p}_i\rangle_{i\in I}\}»\langle\tilde{\overset{\dagger}{n}}\rangle] \mid \lfloor\,\rfloor\circ[\mathbb{G}''«\{S''_i\langle\bar{m}_i\rangle_{i\in I}\}»\langle\tilde{\overset{\dagger}{p}}\rangle]) \approx_r \lfloor\,\rfloor\circ[\mathbb{G}«\{S_i\langle\bar{n}_i\rangle_{i\in I}\}»\langle\tilde{\overset{\dagger}{m}}\rangle]$$

$$\text{where } \mathbb{G} \overset{\text{def}}{=} «\tilde{\eta}»(\tilde{\overset{\dagger}{x}})\bigotimes_{i\in I}!(\nu\,\kappa_i)\overset{\dagger}{x}_i(\tilde{\bar{v}})\,\check{\kappa}_i@(J'_i\cup J''_i).\eta_i\langle\tilde{\bar{v}},\hat{\kappa}_i\rangle \quad \text{and} \quad S_i \overset{\text{def}}{=} S'_i.$$

In other words, the effect of composition on the exclusion is to union the corresponding exclusion sets within each choice branch. For different S'_i and S''_i, especially when they give conflict descriptions about the same subset of methods, we may not be able to find such simple form of stand alone $\mathbb{G}$ and S_i to express the composed behaviour, but the underneath principle remains the same. [Zhang01C] and [Zhang01D] have shown that, from the unlock scheduling point of view, the number of S_i types is finite, and the composition effects can also be grouped to finite number of types, which can be useful for compile time reasoning and code optimisation. Figure 6-1 shows some more examples about application of the identity law in compositional object modelling. The example shown in the left digram indicatess that the same effect of this control can be constructed by three different ways: using the empty control E to extend the scope of controller C to $\tilde{m}$, adding the constraint decribed by C to the empty control E; using two independed controllers C and E_0.



$E \succ C \approx_r C \succ E \approx_r C \mid E_0$

$(C_1\langle\tilde{n}_1,\tilde{p}_1\rangle \succ E_1\langle\tilde{n}_2,\tilde{p}_2\rangle\,) \succ C_2\langle\tilde{p},\tilde{m}\rangle \approx_r C\langle\tilde{n},\tilde{m}\rangle$

where $\tilde{n}_2 = \tilde{n} - \tilde{n}_1$, $\tilde{p} = \tilde{p}_1 \cup \tilde{p}_2$.

Figure 6-1

Another proven property of composition is the association law, held for both ground bisimulation ([Zhang98A]) and the responsive bisimulation ([Zhang01C]): $(C_1 \succ C_2) \succ C_3 \approx_g C_1 \succ (C_2 \succ C_3)$ and $(C_1 \succ C_2) \succ C_3 \approx_r C_1 \succ (C_2 \succ C_3)$.

7 Conclusion

This paper has presented the responsive bisimulation and variations in the κ-calculus. For object systems, where the input name clash can be eliminated, the responsive bisimulations are preserved by parallel composition, output name substitution and choice, can even be congruence.

With the responsive bisimulation, we can have a broader and more generic studies on the behaviours of composed concurrent objects, at where existing bisimulations may fail, enable us to establish the theory of composition with elegant properties and the semantic of an extension to concurrent object-oriented programming languages.

Unlike that in the polar π-calculus, the responsive bisimulation in the κ-calculus is splitted into the κ-version, the version with the internal vision of objects, and non-κ-version, the version with external vision. Maintaining a κ-version responsive bisimulation in object components level will guarrantee the non-κ-version responsive bisimulation in the whole object.

References:

[Aksit92] Mehmet Aksit and Lodewijk Bergmans "Obstacles in Object-oriented Software Development", *OOPSLA '92 Conference Proceedings*, volume 27 of ACM SIGPLAN Notices, pages 341-358, New York, October 1992

[Amadio96] Roberto M. Amadio, Ilaria Castellani and Dacide Sangiorgi, "On Bisimulations for the Asynchronous π-calculus", in *Proceedings of CONCUR'96*, LNCS volume 1119, Springer Verlag, 1996

[Holmes97] David Holmes, James Noble, John Potter, "Aspects of Synchronisation", in Christine Mingins, Roger Duke and Bertrand Meyer, editors, *Technology of Object-Oriented Languages and Systems TOOLS 25 - Proceedings of The 25th International Conference TOOLS* (TOOLS Pacific'97), pages 7-18, Melbourne, Australia, November 1997.

[Honda91] Kohei Honda and Mario Tokoro, "An Object Calculus for Asynchronous Communication", in P. America, editor, *ECOOP'91*, LNCS vol 512, pages 133-147, Springer-Verlag, 1991.

[Honda92] Kohei Honda and Mario Tokoro, “On Asynchronous Communication Semantics”, in M. Tokoro, O. Nierstrasz, and P. Wegner, editors, *Object-Based Concurrent Computing 1991*, LNCS vol 612, pages 21-51, Springer-Verlag, 1992.

[Hüttel96] Hans Hüttel and Josva Kleist, “*Objects as mobile processes*”, Aalborg University, August 1996. URL: http://www.cs.auc.dk/~kleist/ObjMobile

[Jones93] Cliff B. Jones, “A π-calculus Semantics for an Object-based Design Notation”, in E. Best, editor, *Proceedings of CONCUR’93*, volume 715 of *Lecture Notes in computer Science*, pages 158-172. Springer Verlag, 1993

[Liu97] Xinxin Liu and David Walker, “Concurrent Objects as Mobile Processes”, to be appeared in G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press.

[McHale94] Ciaran McHale, “Synchronisation in Concurrent, Object-oriented Languages: Expressive Power, Genericity and Inheritance”, PhD. Thesis, Department of Computer Science, Trinity college, University of Dublin, Ireland, October 1994. URL: ftp://ftp.dsg.cs.tcd.ie/pub/doc/dsg-86.ps.gz

[Milner92] Robin Milner, Joachim Parrow, David Walker, “A Calculus of Mobile Process” (Parts I and II), *Journal of Information and Computation*, 100:1-77, September 1992. URL: http://www.dcs.ed.ac.uk/lfcsreps/EXPORT/89

[Milner92b] Robin Milner and Davide Sangiorgi, “Barbed Bisimulation”, in W. Kuich, editor, *Proceeding of 19th ICALP*, volume 623 of *Lecture Notes in computer Science*, Springer Verlag, 1992

[Milner96] Robin Milner, “*The π-calculus*”, hand-written tutorial. Computer Science Tripos, Cambridge University 1996

[Merro98] Massimo Merro and Davide Sangiorgi, “On Asynchrony in Name-passing calculi”, In 25th ICALP, volume 1442 of *Lecture Notes in computer Science*, pages ??. Springer Verlag, 1998

[Merro00] Massimo Merro, “Locality and Polyadicity in Asynchronous Name-passing Calculi”, In *Proceedings of FOSSACS 2000*, Berlin, Germany, volume 1784, pages 238-251, Lecture Notes in Computer Science, Springer Verlag, 2000

[Nestmann96] Uwe Nestmann and Benjamin C. Pierce, “Decoding Choice Encodings”, *Journal of Information & Computation*, 163: 1-59, November 2000. URL: http://www.brics.dk/RS/99/42

[Noble00] James Noble and John Potter, “Exclusion for Composite Objects”, In *Proceedings of OOPSLA 2000*, Minneapolis, Minnesota USA, ACM press, 2000

[Odersky95a] Martin Odersky, “Polarized Name Passing”, in Proceedings of 15th Foundations of Software Technology and Theoretical Computer Science (FST&TCS'95), Bangalore, India, December 18-20, 1995. URL: http://lampwww.epfl.ch/~odersky/papers

[Odersky95c] Odersky, M. “Polarized bisimulation”, In *Proceedings of Workshop on Logic, Domains, and Programming Languages*, Darmstadt, Germany, 1995

[Philippou96] Anna Philippou and David Walker, “On Transformations of Concurrent-Object Programs”, *Theoretical Computer Sciences*, to appear. Extended abstract in Proceedings of CONCUR'96, papers 131-146, Springer 1996

[Philippou97] Anna Philippou and David Walker, “*A Process-Calculus Analysis of Concurrent Operations on B-Trees*”, Technical report, University of Warwick, UK, 1997

[Pierce95] Benjamin C. Pierce, David N. Turner, “Concurrent Objects in a Process Calculus”, In Takayasy Ito and Akinori Yonezawa, editors, *Theory and Practice of Parallel Programmin (TPPP)*, LNCS 907, pages 187-215. Springer, April 1995. URL: http://www.cis.upenn.edu/~bcpierce/papers

[Pierce96] Benjamin C. Pierce, David N. Turner, “PICT: A Programming Language Based on the π-calculus”. URL: http://www.cis.upenn.edu/~bcpierce/papers

[Ravara97] António Ravara and Vasco T. Vasconcelos, Behavioural types for a calculus of concurrent objects. In C. Lengauer, M. Griebl, and S. Gorlatch, editors, *Proceddings of 3rd International Euro-Par Conference*, LNCS 1300, pages 554--561. Springer-Verlag, 1997

[Sangiorgi92a] David Sangiorgi, “From π-calculus to Higher-Order π-calculus, and Back”, In *Proceedings of TAPSOFT’93*., LNCS 668, Springer Verlag, 1992. URL: http://www-sop.inria.fr/mimosa/personnel/Davide.Sangiorgi/mypapers.html

[Sangiorgi92b] David Sangiorgi, “*Expressing Mobility in Process Algebras: First-Oreder and Higher-Order paradigms*”, PhD thesis, Computer Science Department, University of Edinburgh, UK, 1992. Available from URL: http://www-sop.inria.fr/mimosa/personnel/Davide.Sangiorgi/mypapers.html

[Sangiorgi95] David Sangiorgi, "*Lazy functions and mobile processes*", INRIA Technical Report RR-2515, August 1996. URL: http://www-sop.inria.fr/mimosa/personnel/Davide.Sangiorgi/mypapers.html

[Sangiorgi96] David Sangiorgi, "*An Interpretation of Typed Objects into Typed π-calculus*", INRIA Technical Report RR-3000, August 1996. URL: http://www-sop.inria.fr/mimosa/personnel/Davide.Sangiorgi/mypapers.html

[Sangiorgi96b] David Sangiorgi, "Locality and Non-interleaving Semanitics in Calculi for Mobiule Processes", *Theoretical Computer Science*, 155:39-83, 1996

[Sangiorgi97] David Sangiorgi, "The Name Discipline of Uniform Receptiveness", In 24th ICALP, volume 1256 of *Lecture Notes in computer Science*, Springer Verlag, 1997

[Schneider97] Jean-guy Schneider and Markus Lumpe, "Synchronizing Concurrent Objects in the ", Proceedings of Langages et Modèles à Objets '97, Roland Ducournau and Serge Garlatti (Ed.), Hermes, Roscoff, October 1997, pp. 61-76.

[Walker95] David Walker, "Objects in the π-Calculus", *Information and Computation*, 116(2): 253-271 (1995)

[Zhang97] Xiaogang Zhang and John Potter, "Class-based models in π-calculus", in Christine Mingins, Roger Duke and Bertrand Meyer, editors, *Technology of Object-Oriented Languages and Systems*, *TOOLS 25* (TOOLS Pacific'97), Melbourne, Australia, 24th-27th November 1997, pages 238-251, IEEE Computing Society Press, 1998. URL: ftp://ftp.mpce.mq.edu.au/pub/mri/people/xzhang/papers/class97.ps.gz

[Zhang98A] Xiaogang Zhang and John Potter, "*Compositional Concurrency Constraints for Object Models in π-calculus*", Technical Report C/TR-9804, Macquarie University, Sydney, Australia, 1998. URL: ftp://ftp.mpce.mq.edu.au/pub/mri/people/xzhang/papers/TR98-04.doc

[Zhang98B] Xiaogang Zhang and John Potter, "A Compostion Approach to Concurrent Objects", in Jian Chen, Mingshu Li, Christine Mingins and Bertrand Meyer, editors, *Technology of Object-Oriented Languages and Systems*, *TOOLS 27* (TOOLS Asia'98), Beijing, China, 22nd-25th September 1998, pages 116-126, IEEE Computing Society Press, 1998. URL: ftp://ftp.mpce.mq.edu.au/pub/mri/people/xzhang/papers/tools27.ps.gz

[Zhang01A] Xiaogang Zhang and John Potter, "*The Responsive Bisimulations in the polar π-calculus*", Technical report UNSW-CSE-TR-0203.

[Zhang01B] Xiaogang Zhang and John Potter, "*A Constraint Description Calculus for Compositional Concurrent Objects*", Technical report UNSW-CSE-TR-0204.

[Zhang01C] Xiaogang Zhang and John Potter, "*Compositional Concurrent Objects*", Technical report of CSE UNSW, Aug. 2001.

[Zhang01D] Xiaogang Zhang and John Potter, "A *Compositional Concurrent Object Model, -- From Theory to Practise*", Technical report of CSE UNSW, Augest 2001, in preparation.