The Responsive Bisimulations in the \( \kappa \)-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 ($\forall n (m \parallel k \cdot n \cdot P)$ and $k \cdot m \cdot P$) 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 maintaining the syntactical simplicity similar to the normal π-calculus, but is still difficult to express general behaviours 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 concurrent 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 \( \kappa \)-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 \( \pi \)-calculus expression \((\forall n)(\exists m) [k.n.P] \) and \( k.m.P \) then can be achieved. However, in the earlier paper, the responsive bisimulation was described in the polar \( \pi \)-calculus, which added a few improved features for modelling concurrent objects while maintaining the syntactical simplicity similar to the normal \( \pi \)-calculus, but is still difficult to express general behaviours of concurrent objects efficiently. The \( \kappa \)-calculus, where locks are included as primitive, in the other hand, is more expressive and flexible in modelling compositional concurrent objects.

This paper presents responsive bisimulation in the \( \kappa \)-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 \( \pi \)-calculus ([Milner92], [Milner96]) and its variations have been applied to modelling concurrent object systems ([Walker95], [Jones93], [Sangiorg96], [Hütel96], [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 \( \pi \)-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 \( F = \Pi [n_1(x), M_1(x), \ldots, n_k(x), M_k(x)] \), 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 \( F_B = [n_1(x), M_1(x)] ; n_2(x), M_2(x) \), where \( n_1(x), M_1(x) \) and \( n_2(x), M_2(x) \) 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 \( C_B = m(x), \bar{n}(x) + m(x), \bar{n}(x) \), where the sum operator in fact represents a mutual exclusion lock on those methods. Then the parallel composition of the two processes, \( (\forall n)(C_B ; F) \), will be weakly bisimilar to \( R = [m(x), M(x), \bar{n}(x) + m(x), M(x)] \), 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 \( (\forall n)(m \bar{n} | n.P) \) 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 $\kappa$ (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 $\kappa$, 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.

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 $O_1\mid R$ and $O_2\mid R$ are not weakly barbed-bisimilar for some $R$, such as $R \equiv m(a)$.

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 $C \triangleright F$ 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., $C \triangleright (C_1 \triangleright C_2) \equiv (C_1 \triangleright C_2) \triangleright F$, but also the identity law, i.e., there is some empty control (identity) $E$ such that for all $F$ satisfying $E \triangleright F$, the composed object $E \triangleright F$ is equivalent to the original object $F$, and for all control process $C$ satisfying either $E \triangleright C$ or $C \triangleright E$, the three control processes $E \triangleright C$, $C \triangleright E$ and $C$ are all equivalent ([Zhang01C]).

In [Zhang01A] the responsive bisimulation was studied using the polar $\pi$-calculus as the mathematical tool, which adopts the concept of polarised names from [Odersky96a], and then syntactically 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 exclusion relations between object methods are difficult to be presented efficiently and compositionally in the $\pi$-calculus and most variations, including the polar $\pi$-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 $\pi$-calculus respectively as the two control processes $\phi \equiv m_1(x) . n_1(x) + m_2(x) . n_2(x)$ and $\phi \equiv m_1(x) . n_1(x) \mid m_3(x) . n_3(x)$.
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., \( C \equiv m_2(\mathcal{X}) \cdot m_3(\mathcal{X}) \mid !m_2(\mathcal{X}), !m_3(\mathcal{X}) \);  
2) Mutually exclusive should be maintained between method \( m_2 \) and \( m_3 \), i.e., \( C \equiv m_2(\mathcal{X}) \cdot m_3(\mathcal{X}) + m_3(\mathcal{X}), !m_2(\mathcal{X}) \);  
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: \( C \equiv (m_1(\mathcal{X}), m_2(\mathcal{X}), m_3(\mathcal{X})) \mid !m_2(\mathcal{X}), !m_3(\mathcal{X}) \). However, it is difficult for 2) and 3), because:

a) The entire control \( C \) 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 \( m_1 \times m_2 \), \( m_1 \times m_3 \), \( m_2 \times m_3 \), and \( m_1 \times m_2 + m_1 \times m_3 \).

However, the algebra of exclusion is static expressible, unable to present the dynamical behaviour of concurrent objects. To solve this problem, [Zhang01B] proposed an extended calculus, the \( \kappa \)-calculus which welds the mobility power of the \( \pi \)-calculus with the synchronisation expressiveness of the algebra of exclusion ([Noble00]). With the \( \kappa \)-calculus, a control \( C \) will have the form \( C \equiv \langle S \rangle \cdot \langle \mathcal{F} \rangle \cdot \langle \mathcal{G} \rangle \), where \( \mathcal{F} \) specifies the exclusion relations, and each \( S_i \) gives some scheduling information such as unlocking or early return on the \( i \)-th method. As the example, for each of the previous mentioned three situations we may write the \( \mathcal{F} \) as:

1) \( \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \odot \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \odot \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \odot \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \);  
2) \( \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \odot \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \odot \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \);  
3) \( \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \odot \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \odot \langle \mathbf{v} \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \mathbf{m}(\mathcal{X}) \cdot \eta(\mathcal{X}, \mathcal{X}) \rangle \).

As we can see here, the \( \kappa \)-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 \( \mathcal{F} \).

In this paper we put the two pieces together, present the responsive bisimulation in the \( \kappa \)-calculus, and therefore form a full base for studying of compositional concurrent objects and the theory of composition.

The rest of the paper is structured as follows: section 2 briefly introduces the \( \kappa \)-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 \( \kappa \)-calculus

The \( \kappa \)-calculus ([Zhang01B]) is a process calculus especially suitable for modelling the composition behaviours of concurrent objects. Like the asynchronous \( \pi \)-calculus ([Amadio96]), it uses asynchronous communication, i.e., an output action does not block other actions. Like the polar \( \pi \)-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 \( \kappa \)-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 \( \pi \)-calculus while providing more power on behaviour separation.

The major significance in the \( \kappa \)-calculus is the inclusion of lock as primitive. In the conventional CCS or \( \pi \)-calculus, 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 \( \kappa \)-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 \( \kappa \)-calculus to handle the expressivity of the algebra of exclusion ([Noble00]) for methods of exclusion of concurrent objects, allowing the separation of some major concurrently behaviours of objects to be presented in a much more natural and cleaner way. The \( \kappa \)-calculus distinguishes the labels for communication channel names and that for locking keys, in order to prevent cross using by mistake.

2.1 The syntax of the \( \kappa \)-calculus

In the \( \kappa \)-calculus we distinguish two disjoint sets of label names, the communication channel names, and the key names for locking. Let \( M \) be the set of all communication channel names, ranged over by expressions \( m, n, \bar{n} \) and variables \( x, y \). Let \( \bar{M} \equiv \{ m \in M \} \) and \( \bar{M} \equiv \{ m \in M \} \) be the sets of input polar and output polar of all channel names respectively. Let \( \bar{\kappa} \) be set of all release keys of locking, ranged over by \( \kappa \). Let \( \bar{\kappa} \equiv \{ \kappa \in \kappa \} \) and \( \bar{\kappa} \equiv \{ \kappa \in \kappa \} \) be the sets of input polar and output polar of all keys respectively. Then the set of all label names is \( \bar{\kappa} \equiv \bar{M} \cup \bar{\kappa} \) ranged over by \( n \). Consequently, we have various sets of polar, such as \( \bar{M} \equiv \bar{M} \bar{\kappa} \equiv \bar{M} \bar{\kappa} \equiv \bar{M} \bar{\kappa} \equiv \bar{M} \bar{\kappa} \). Let \( a, b \in \bar{\kappa} \) be polar constants, and \( w \in \bar{\kappa} \) be polar variables. Let both \( \bar{r} \) and \( \bar{r} \), where \( I \) is an index set of arity \( n \), be abbreviations for \( r_1 \ldots r_n \).

The generic process terms \( P \) in the \( \kappa \)-calculus are generated by the following grammars:

\[
P ::= P_0 | m(\tilde{n})P | P_1 \perp P_2 | A(G) | A(\bar{a}) | \eta | G ::= B | (v\tilde{n})G | G_1 \otimes G_2 | D(\bar{a}) | \eta \tilde{P} | P[\bar{b}]P | \{ (v\kappa)\beta \} P
\]

A ::= \( (\tilde{\nu}G) \), \( \bar{a} \equiv (\tilde{\nu}G) \), \( \beta ::= m(\tilde{\nu})L \), \( J ::= \bar{r} \).

The set of all actions a process may take can be specified by \( \alpha ::= m(\tilde{n}) | (v\bar{b})m(\bar{r}) | \bar{a} | \beta \). where \( \bar{b} \equiv \bar{\nu} \) and \( \bar{m} \equiv \bar{\nu} \).

Most process terms (\( P \)-terms) are similar to those in normal \( \pi \)-calculus; \( P_0 \) is the inactive (terminated) process; \( m(\tilde{n}) \) is the output action which sends out polar \( \tilde{n} \) into the channel \( m \); \( (v\tilde{n})P \) binds the set of labels \( \tilde{n} \), and therefore both polars of each of them, within the scope of \( P \); \( P_1 \perp P_2 \) indicates two processes run in parallel, and \( A(\bar{a}) \) is an instance of parameterised process agent, giving the process agent abstraction \( A \equiv (\tilde{\nu}G) \) obeying \( (A(\tilde{n})P)(\bar{a}) = P(\bar{a}P0) \). \( \eta \) is a process variable. For the rest two terms, \( \bar{a} \) is the action which emits the unlock signal within the scope where the name \( \kappa \) is bound; and \( A(G) \) is the guarded exclusive choice (GEC choice), where \( G \) defines the exclusion behaviour which can place some locks on the process itself, and \( A \) records the lock status. For the choice terms (\( G \)-terms), \( B \) is a choice branch; \( G_1 \otimes G_2 \) is the choice composition; \( (\tilde{\nu}G) \) and \( D(\bar{a}) \) are abstraction and instance of choice agent respectively, obeying \( (\tilde{\nu}G)(\tilde{x}) = G(\tilde{x}) \); higher order \( G \)-term agent \( \tilde{m} \equiv (\tilde{\nu}G) \) accepts processes as parameters in the double angled brackets, and obeying \( (\tilde{\nu}G) \equiv G(\tilde{x}) \). \( \kappa_0 \) is the unreachable choice, in the future we can omit the subscript of both \( \kappa_0 \) and \( 0 \), without any ambiguity. Unlike that in \( \pi \), every branch \( B \) here always behaves as a (lazy) replication, among them, \( "(v\kappa)" \) creates a fresh key \( \kappa \) private to each replicated copy; in \( \beta P \) the action prefix operator \( "\cdot P " \) indicates the execution of action \( \beta \) before the execution of the continuation process \( P \); the action \( m(\tilde{x})L \), where we stipulate that \( \{ \tilde{x} \} \cap \kappa(\tilde{L}) = 0 \), produces two simultaneous events: receiving information \( \tilde{x} \) from the input port of channel \( m \), and triggering the lock \( L \); the lock \( L = \kappa_0 \) read as "lock all input channels in \( J \) with key \( \kappa_0 \)", where the exclusion set \( J \) specifies the channels to be locked within the GEC choice and \( \kappa \) is the key for unlocking the lock; abbreviation \( L = (v)\kappa \) indicates a lock with an anonymous key, that is, \( m(\tilde{x})(v)\kappa J \lor P = (v)\kappa \tilde{m}(\tilde{x}) \kappa_0 J \). For \( \kappa \in J(P) \), in other words, it is an unreachable lock; \( M \) is the entire \( M \) 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, \( A \), acts as a state machine maintaining and monitoring the current status of locks, and is described in an independent language. Different \( A \) 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 a single lock ([Zhang01]), is defined the grammar \( A ::= \{ J \} \equiv \{ J \} \equiv A \) and the structural equivalencies rules are shown in Figure 2-1.

![Figure 2-1 Structural equivalence of locking status terms](image-url)
Notation 2-1: Some auxiliary operations/functions (the formal definitions can be found in [Zhang01B]) are needed for integrating an $A$ language into the $\kappa$-calculus:

- $\text{guard}(G)$: gives the set of all branches' input prefix channel names in $G$, and defined by
  \[ \text{guard}(\text{bn}(\exists)LP) \equiv \text{bn}; \quad \text{guard}(\text{in}(\text{bn})G) \equiv \text{guard}(G); \quad \text{guard}(G \otimes G) \equiv \text{guard}(G) \cup (\text{guard}(G)); \]
- $\text{lock}(\kappa, A)$: gives the truth value for whether $A$ indicates all the input polar(s) appeared in $J$ are locked by $\kappa$;
- $\text{lock}(A)$: gives the set of all channel names for which their input polar(s) are indicated by $A$ as locked;
- $\text{key}(A)$: gives the set of all key $\kappa$ for which there exists some $J \otimes \emptyset$ such that $\text{lock}(\kappa, A) \equiv \text{true}$;
- $\text{add}(L, A)$: gives the new locking status after adding $L$ to the original locking status $A$;
- $A/L$: gives the new locking status after removing $L$ from the original locking status $A$.

We usually use $A \equiv \emptyset$ to represent an empty lock, and for any $A$ language, we always require that:

- $\text{lock}(\kappa, \emptyset) \equiv \text{false}$, $\text{lock}(\emptyset) \equiv \emptyset$, $\text{key}(\emptyset) \equiv \emptyset$, $\text{add}(L, \emptyset) \equiv \emptyset$ and $\emptyset \cap \emptyset \equiv \emptyset$.

Notation 2-2: If $m \in \text{lock}(A)$, we say that $A$ allows the commitment on $m$, denoted as $A \downarrow m$;

- if $m \in \text{lock}(A)$, we say that $A$ blocks the channel $m$, denoted as $A \downarrow m$.

If for some $J \otimes \emptyset, J \otimes \emptyset$ and $\text{lock}(\kappa, A)$, we say that $A$ can commit $\text{lock}(J)$, denoted as $A \downarrow \text{lock}(J)$;

- otherwise we say that $A$ cannot commit $\kappa$ over $J$, denoted as $A \downarrow \text{lock}(J)$. 

If for some $J \otimes \text{lock}(A), A \downarrow \text{lock}(J)$, we say that $A$ can commit $\kappa$ over $J$,

- otherwise we say that $A$ cannot commit $\kappa$ over $J$, denoted as $A \downarrow \text{lock}(J)$.

In the form of labelled transition, we denote $A \downarrow \text{lock}(J)$ for $m \in \text{lock}(A)$ and $A = \text{add}(L, A)$; and $A \downarrow \text{lock}(J)$ for $A \downarrow \text{lock}(J)$ and $A = \text{lock}(A)$.

Notation 2-3: Similar to the polar $\pi$-calculus, besides the functions \text{in}, \text{bn} and \text{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 \text{fin}, \text{bin}, \text{in}, \text{fin}, \text{bin} and \text{on} to identify free, bound and all input or output polar(s). Further more, as in the $\kappa$-calculus we distinguish communication channel names and keys, we also use finer grained functions, \text{fin}, \text{bin}, \text{n}, \text{fin}, \text{bin}, \text{in}, \text{fin}, \text{bin}, \text{on} and \text{on} for communication channels only, and \text{fin}, \text{bin}, \text{n}, \text{fin}, \text{bin}, \text{in}, \text{fin}, \text{bin}, \text{on} and \text{on} for keys only.

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

- $\text{nm}(\emptyset).P \equiv \emptyset \cup \{ \text{nm}(\emptyset)(v) = \text{nm}(P) \}$,
- $\prod \text{nm}(\emptyset).P = \prod \{ \text{nm}(\emptyset)(v) = \text{nm}(P) \}$,
- $\Sigma \text{nm}(\emptyset).P \equiv \Sigma \{ \text{nm}(\emptyset)(v) = \text{nm}(P) \}$,
- $\prod \text{nm}(\emptyset).P = \prod \{ \text{nm}(\emptyset)(v) = \text{nm}(P) \}$.

These abbreviations give an illustration of that for input-prefixed processes, the standard parallel and sum compositions in conventional $\pi$-calculus become special cases of GEC choice in the $\kappa$-calculus. Further more, as these abbreviations suggested, encoding a polar $\pi$-calculus term into the $\kappa$-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 $\pi$-calculus can be considered as a sub-calculus of the $\kappa$-calculus.

2.2 The semantics of the $\kappa$-calculus

The structural equivalences and labelled transitions in the $\kappa$-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 $\pi$-calculus, we can see that:

1. an input action $\text{nm}(\emptyset)$ invokes a new copy of continuation process $P$ from a GEC choice and triggers a lock $L$ which may change $A$, the locking state of GEC choice, an unlock signal $\kappa$ may also change the locking state $A$, 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: | $P_1 \mid 0_p \equiv P_1$ | $G_1 \otimes 0_G \equiv G_1$ |
| str-SUM2: | $P_1 \mid P_2 \equiv P_1 \mid P_1$ | $G_1 \otimes G_2 \equiv G_2 \otimes G_1$ |
| str-SUM3: | $P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_3) \mid P_3$ | $G_1 \otimes (G_2 \otimes G_3) \equiv (G_1 \otimes G_2) \otimes G_3$ |

### Null

| str-NUL: | $\lambda \Gamma (0_G) \equiv 0_P$ |

### Instance

| str-INS: | $(\lambda \Gamma)(\lambda \alpha) = \Gamma[\alpha \rightarrow \lambda \alpha]$ | $(\lambda \Gamma)(\lambda \alpha) = \Gamma[\alpha \rightarrow \lambda \alpha]$ |

### Scope

| str-SCP1: | $(\forall \Gamma)(\forall n P \equiv P, \text{ if } n \notin \beta(P))$ | $(\forall \Gamma)(\forall n G \equiv G, \text{ if } n \notin \beta(G))$ |
| str-SCP2: | $(\forall \Gamma)(\forall n_2 P \equiv (\forall n_2)(\forall n_1) P)$ | $(\forall \Gamma)(\forall n_1) P \equiv (\forall n_1 n_2) P$ |
| str-SCP3: | $(\forall \Gamma)(\forall m (\lambda \alpha) \equiv 0_P)$ | $(\forall \Gamma)(\forall m (\lambda \alpha) \equiv 0_P)$ |
| str-SCP4: | $(\forall \Gamma)(\forall \alpha) P \equiv (\forall \alpha) P[\alpha \rightarrow \lambda \alpha]$, if $\alpha \notin \beta(G)$ | $(\forall \Gamma)(\forall \alpha) P \equiv (\forall \alpha) P[\alpha \rightarrow \lambda \alpha]$, if $\alpha \notin \beta(G)$ |
| str-REN: | $(\forall \Gamma)(\forall n_1) P \equiv (\forall n_2) P[\alpha \rightarrow \lambda \alpha]$, if $n \notin \beta(P)$ | $(\forall \Gamma)(\forall n_1) G \equiv (\forall n_2) G[\alpha \rightarrow \lambda \alpha]$, if $n \notin \beta(G)$ |

---

**Figure 2-2 Structural congruence rules for the $\kappa$-calculus**

| tr-OUT: | $m(\bar{n}) \vdash \Gamma \Rightarrow 0_P$ | $P \vdash \Gamma \Rightarrow \kappa \Rightarrow 0_P$ |
| tr-IN: | $A \vdash \Gamma L \Rightarrow A^\prime$ | $A \vdash \kappa \Rightarrow A'(G)$ |
| tr-RELS: | $A \vdash \kappa L \Rightarrow A'(G)$, where $\Gamma \equiv \gamma(G)$ | $A \vdash \kappa \Rightarrow A'(G)$ |
| tr-SYN1: | $(\forall m(P \mid Q \Rightarrow \kappa \mid \kappa \Rightarrow Q))$ | $(\forall m(Q \mid P \Rightarrow \kappa \mid \kappa \Rightarrow Q))$ |
| tr-RES: | $(\forall m(P \mid \bar{n} \Rightarrow \kappa \mid \kappa \Rightarrow Q))$ | $(\forall m(Q \mid P \Rightarrow \kappa \mid \kappa \Rightarrow Q))$ |

---

**Figure 2-3 Labelled transition rules for process terms in the $\kappa$-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 $n \notin \beta(P)$ may be picked up automatically so that the process $(\forall n_1) (\forall \bar{n} n_1) (\forall n_2) P[\alpha \rightarrow \lambda \alpha]$ can be used to replace $(\forall n_1) (\forall \bar{n} n_1) (\forall n_2) P[\alpha \rightarrow \lambda \alpha]$ without mention.

**Remark 2-5:** Similar to the polar $\pi$-calculus, in the the $\kappa$-calculus the $\tau$ action is truly internal, that is, neither visible nor interruptible by external observers. Therefore, the name restrictions in rule tr-SYN1 and tr-SYN2 are required. Without it, the synchronisation will not be considered as an internal action, but a two steps action,
such as $P \mid Q \overrightarrow{\alpha} \overleftarrow{\nu}(\overrightarrow{\alpha})Q \Leftrightarrow P \mid Q \overrightarrow{\alpha} \overleftarrow{\nu} P \mid Q$, where both steps are visible for external observers. This strong requirement on $\tau$ actions is necessary for guaranteeing the standard rule $\beta(\tau) = \beta(\eta) = \emptyset$ ([Amadio96]) valid, and is necessary for preserving $\tau$ actions in output polaris substitution.

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

**Definition 2-6:** $P \overrightarrow{\alpha} P'$ iff $P (\overrightarrow{\alpha})^* P'$;  $P \overrightarrow{\alpha} \overleftarrow{\nu} P'$ iff $P \overrightarrow{\alpha} \overleftarrow{\nu} \overrightarrow{\alpha} \overleftarrow{\nu} P'$, where $\alpha \neq \tau$.

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

**Definition 2-7:** $P \overrightarrow{\nu} P'$ iff $(\forall m) P \overrightarrow{\nu} (\forall m) P'$ for some $m$;  $P \overrightarrow{\nu} P'$ iff $(\forall m) P \overrightarrow{\nu} (\forall m) P'$ for some $m$.

Clearly, $P \overrightarrow{\nu} P'$ implies $P \overrightarrow{\nu} P'$, and $P \overrightarrow{\nu} P'$ implies $P \overrightarrow{\nu} P'$, and therefore a variant of the rule tr-SYNC1 can be written as: if $P \overrightarrow{\nu} P'$ and $Q \overrightarrow{\nu} Q'$, then $P \overrightarrow{\nu} P'$. 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 $\alpha$, denoted as $P \mid \alpha$, if there exists some $P'$ such that $P \overrightarrow{\alpha} P'$.
- Process $P$ can commit on input polar $\overleftarrow{\nu} m$, denoted as $P \mid \overleftarrow{\nu} m$, if there exists some input action $\alpha = m(\overleftarrow{\nu})$ s.t. $P \mid \alpha$.
- Process $P$ can commit on output polar $m$, denoted as $P \mid \overrightarrow{\nu} m$, if there is some output action $\alpha = m(\overrightarrow{\nu})$ s.t. $P \mid \alpha$.
- Process $P$ can commit the action sequence $\overrightarrow{\alpha}$, denoted as $P \overrightarrow{\alpha}$, if $P \overrightarrow{\alpha} \overrightarrow{\alpha} \overrightarrow{\alpha} P'$ or as an abbreviation, $P \overrightarrow{\alpha} P'$.

The weak commitments $\overrightarrow{\nu}$, is obtained by replacing $\overrightarrow{\nu}$ with $\overrightarrow{\nu}$ and $\overrightarrow{\nu}$ with $\overrightarrow{\nu}$ through out.

**Definition 2-9:** Process $P$ is a derivative of process $P$, if there exists some finite sequence $\nu$ such that $P \overrightarrow{\nu} P'$.

### 3 Responsive bisimulation in the $\kappa$-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 $\kappa$-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 $\pi$-calculus;
2. take locking/releasing actions into account and therefore produce the "$\kappa$-variation", an even finer version, for each of those bisimulation relations.

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

Generally say, the $\kappa$-version bisimulations are needed for measuring properties of object components, when non-$\kappa$-version bisimulations are interested in measuring overall behaviour of composed objects.

The barded 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 $\kappa$-calculus only output polaris (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 $\equiv$ on $P$-terms is a (strong) barbed bisimulation if whenever $P \xrightarrow{a} Q$ then $P \xrightarrow{a} Q$ for all $a \in M$, and $P \xrightarrow{R} \equiv Q \xrightarrow{R}$ such that $Q \xrightarrow{Q}$ and $P \xrightarrow{Q}$.

Let $\approx_b$ be the largest strong barbed bisimulation. The notion of weak barbed bisimulation $\approx^w_b$ is obtained by replacing everywhere the transition $\downarrow$ with $\circ$ and $\xrightarrow{a}$ with $\xrightarrow{a}$ throughout.

For $\kappa$-versions, the strong and weak **barbed $\kappa$-bisimulation** $\approx_{\kappa b}$ and $\approx_{\kappa b}^w$ respectively, are obtained by extend $\approx_{\kappa b}^w$ in the above definition.

Since 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 $\kappa$-calculus, process context $C_{[.]}$ is given by $C := [.] | (v \cdot) C \big| (C \cdot) C \big| A \cdot \Omega(v \cdot) \kappa \cdot \kappa \cdot G$.

**Definition 3.12:** Let $C_{[.]}$ be process context, then we define the barbed equivalences and their $\kappa$-versions as strong and weak **barbed equivalence**:

- $P \equiv_{\approx_b} Q$ if $\forall C., (C_{[.]}) \approx_{\approx_b} (C_{[.]})$.
- $P \equiv_{\approx} Q$ if $\forall C., (C_{[.]}) \approx_{\approx} (C_{[.]})$.

Strong and weak **barbed $\kappa$-equivalence**:

- $P \equiv_{\approx_{\kappa b}} Q$ if $\forall C., (C_{[.]}) \approx_{\approx_{\kappa b}} (C_{[.]})$.
- $P \equiv_{\approx_{\kappa}} Q$ if $\forall C., (C_{[.]}) \approx_{\approx_{\kappa}} (C_{[.]})$.

On for the still weaker versions similar in [Amadio01], let $R$ be arbitrary process, then we define that strong and weak **barbed $1$-equivalence**:

- $P \equiv_{\approx_{\kappa b}} Q$ if $\forall R. (R \Rightarrow P \Rightarrow Q)$.
- $P \equiv_{\approx_{\kappa}} Q$ if $\forall R. (R \Rightarrow P \Rightarrow Q)$.

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 $\kappa$-calculus as $O_1 \approx_{\kappa b} (v \cdot) (m \cdot) (\kappa \cdot) (\cdot \cdot) \Rightarrow (\cdot \cdot) \Rightarrow (\cdot \cdot)$ and $O_2 \approx_{\kappa b} (\kappa \cdot) (\cdot \cdot) \Rightarrow (\cdot \cdot) \Rightarrow (\cdot \cdot)$ 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:

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 $m(p)$, then we have

\[ O_1 \downarrow m(p) \Rightarrow Q_1 \text{ and } O_2 \downarrow m(p) \Rightarrow Q_2 \text{, where } Q_1 \approx_{\kappa b} (v \cdot) (m \cdot) (\kappa \cdot) (\cdot \cdot) \Rightarrow (\cdot \cdot) \Rightarrow (\cdot \cdot) \text{ and } Q_2 \approx_{\kappa b} (\kappa \cdot) (\cdot \cdot) \Rightarrow (\cdot \cdot) \Rightarrow (\cdot \cdot). \]

Since $Q_1 \downarrow m$ while $Q_2 \downarrow m$, therefore $O_1 \downarrow m(p) \approx_{\kappa b} O_2 \downarrow m(p)$, that is, $O_1 \approx_{\kappa b} O_2$.

Second, it cannot prevent input names clash between the testing environment and the processes being tested. For example, let $R \approx m(x) \cdot x(q) \cdot m(p)$, then as shown in Figure 3-1, $O_1 \downarrow R$ can take two different reduction paths:

- either $O_1 \downarrow R \Rightarrow (v \cdot) (m \cdot) (\kappa \cdot) (\cdot \cdot) \Rightarrow (\cdot \cdot) \Rightarrow (\cdot \cdot)$ or $O_1 \downarrow R \Rightarrow (\cdot \cdot)$, while $O_2 \downarrow R$ has only one reduction path, $O_2 \downarrow R \Rightarrow O_2 \downarrow p(q)$. Therefore $O_1 \downarrow R \approx_{\kappa b} O_2 \downarrow R$, that is $O_1 \approx_{\kappa b} O_2$.

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 synchronisations which have delay on inputting 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  or-bisimulation, defined in the same way as that in [Zhang01A], can provide this degree of observation:

**Definition 3-13:** The (strong)  or-bisimulation is a symmetric relation for processes such that whenever $PSQ$ then $P \xrightarrow{a} P'$ implies $Q \xrightarrow{a} Q'$ and $P \xrightarrow{a} Q$ for all action $a$ in the form of either $a=(v,\overline{v})m(\overline{u})$ or $a=\tau$, and $b(\alpha)(v,\overline{v})Q=\emptyset$.

The  or-bisimulation, (strong)  or-bisimulation, is a strong or-bisimulation such that whenever $PSQ$ then $P \xrightarrow{a} P'$ implies $Q \xrightarrow{a} Q'$ and $P \xrightarrow{a} Q$ for all $a \in \mu K^L$.

The weak or-bisimulation and weak  or-bisimulation are obtained by replacing $\mu K$ with $\mu K$ everywhere above respectively. We denote $\approx_{or}$ be the largest or-bisimulation, and $\approx_{or}$ be the largest weak or-bisimulation, $\approx_{or}$ be the largest  or-bisimulation, and $\approx_{or}$ be the largest weak  or-bisimulation.

**Lemma 3-14:** Each of  or-bisimulations, $\mathbb{S}$, is preserved by restriction, that is, $PSQ$ implies $(v \ni P)P (v \ni Q)$.

**Proof:** This can be proven by show that $R \subseteq \{(v \ni P), (v \ni Q) : PSQ\}$ is a $\mathbb{S}$. Here we only give the proof for the strong  or-bisimulations, $\subseteq \subseteq \subseteq$ all others can be proven similarly. Assume $(v \ni P) \xrightarrow{a} P'$ for some arbitrary action $a$, where $a$ is not a communication input action (i.e., $a \notin \mu m(\overline{u})$), then it is only possible in one of the following two cases:

1. $(v \ni P) \xrightarrow{\overline{a}} (v \ni P')$. By rule tr-RES, $(v \ni P) \xrightarrow{\overline{a}} (v \ni P')$, so $P = (v \ni P')$. By $PSQ$, we have $Q \xrightarrow{a} Q'$ and $P \xrightarrow{a} Q$. By tr-RES, $(v \ni Q) \xrightarrow{a} (v \ni Q')$ and we have $((v \ni P), (v \ni Q) \in R$.

2. $a$ is an output action of the form $a=(v,\overline{v})m(\overline{u})$ where $m \ni v$ and $v \xrightarrow{\overline{a}} (v \ni v) \xrightarrow{\overline{a}} v$, and $P \xrightarrow{a} (v \ni P')$. By $PSQ$, we have $Q \xrightarrow{a} Q'$ and $P \xrightarrow{a} Q'. Let \xrightarrow{\overline{a}} \xrightarrow{\overline{a}}$, by rule str-SCP2 and str-SUM2, $(v \ni P) = (v \ni v)(v \overline{v} P)$ and $(v \ni Q) = (v \ni v)(v \overline{v} Q)$. By the tr-OUT, we got $(v \ni v)(v \overline{v} P) \xrightarrow{a} (v \overline{v} Q)$ and $(v \ni v)(v \overline{v} Q) \xrightarrow{a} (v \overline{v} Q)'$, however $((v \ni v)(v \overline{v} P), (v \ni v)(v \overline{v} Q) \in R$.

By the definition of  or-bisimulations, $\mathbb{S}$, we have $R \subseteq \mathbb{S}$.

The or-bisimulations gives a measurement on processes states by observing available reductions and output actions, but can not determine how a process responds 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 $[m(\overline{u})]P$, the *localisation* of the sent message $m(\overline{u})$ with process $P$, into the process syntax. Properties for this term are shown in Figure 3-2.

<table>
<thead>
<tr>
<th>Structural equivalence:</th>
<th>Transition:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Str_NULL</strong> $[m(\overline{u})]0 = 0$;</td>
<td><strong>ITr_SYNC3</strong> $[m(\overline{u})]P \xrightarrow{a} [m(\overline{u})]P$, $[m(\overline{u})]P \xrightarrow{a} [m(\overline{u})]P$;</td>
</tr>
<tr>
<td><strong>Str_LOC</strong> $(v m) [m(\overline{u})]P = (v m) [m(\overline{u})]P$;</td>
<td><strong>ITr_INV</strong> $[m(\overline{u})]P \xrightarrow{a} [m(\overline{u})]P$, $[m(\overline{u})]P \xrightarrow{a} [m(\overline{u})]P$;</td>
</tr>
<tr>
<td><strong>Str_IND</strong> $([m(\overline{u})]P</td>
<td>Q) = [m(\overline{u})]([P</td>
</tr>
</tbody>
</table>

**Figure 3-2** Localised output action.

The term $[m(\overline{u})]P$ couples $P$ with the message $\overline{u}$ which is buffered in channel, and unobservable from outside, even though the output polar $m$ may have been known by outsiders. We may consider the difference between $[m(\overline{u})]P$ and $[m(\overline{u})]P$ as that, in the former the $m(\overline{u})$ is an outgoing message to be buffered into the channel $m$, while in the latter, $[m(\overline{u})]$ is a buffered message arriving from the channel $m$ and waiting to be picked up by $P$. The $[m(\overline{u})]$ privatises
neither polar \( m \) nor \( m \), but the message \( \tilde{u} \). In other words, the \( \lfloor m(\tilde{u}) \rfloor \) 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 \( m \) rather than output polar \( m \) appears in \( \lfloor m(\tilde{u}) \rfloor P \).

The term \( \lfloor m(\tilde{u}) \rfloor P \) is not for modelling processes, but only designed to express r1-bisimulation relations between processes. In this sense, we may read \( \lfloor m(\tilde{u}) \rfloor P \) 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 \( m \). From this point of view, the using of input polar \( m \) rather than output polar \( m \) is necessary to prevent an input polar substitution, caused by input prefixing, changes the static behaviour of \( P \).

The rule \( \text{tr}_r \text{-SYNC}3 \) added a new case for defining the \( \tau \) action. Unlike in rule \( \text{tr}_r \text{-SYNC}1 \), here is no name restriction is required. However, since only the input polar, \( m \), of the channel name \( m \) is involved, and the reservation of \( \tau \) actions is maintained input prefixing.

**Corollary 3-16:** The following conclusion can be immediately drew from the rules in Figure 3-2:

1. If \( P \text{m}(\tilde{u}) \quad P \) then \( (\text{m} \quad m) \quad m(\tilde{u}) \quad P \).
2. \( P \tau \alpha \) implies \( \lfloor m(\tilde{u}) \rfloor P \tau \alpha \) if \( \alpha \neq \tau \), or, \( \alpha = \tau \) but \( P \tilde{m} \).
3. \( (\lfloor m(\tilde{u}) \rfloor P \tilde{m}) \).

Now we can begin to introduce new behaviour equivalence relations.

**Definition 3-17:** Let \( \mathcal{T}[\cdot] \) be the **responsive testing context** of syntax \( \mathcal{T}[\cdot] \) \( \lfloor m(\tilde{u}) \rfloor \), then we define the strong and weak **responsive equivalence**: \( \cong Q \) if \( \forall \mathcal{T}(\mathcal{T}[P] \cong \mathcal{T}[Q]) \), \( \equiv Q \) if \( \forall \mathcal{T}(\mathcal{T}[P] \equiv \mathcal{T}[Q]) \); the strong and weak **\kappa_r**-equivalence: \( \cong_{\kappa_r} Q \) if \( \forall \mathcal{T}(\mathcal{T}[P] \cong_{\kappa_r} \mathcal{T}[Q]) \), \( \equiv_{\kappa_r} Q \) if \( \forall \mathcal{T}(\mathcal{T}[P] \equiv_{\kappa_r} \mathcal{T}[Q]) \).

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) **\( r1 \)-bisimulation** is a strong (or weak, respectively) \( \tau \)-bisimulation \( S \) if whenever \( P S Q \) then \( \lfloor m(\tilde{u}) \rfloor P S \lfloor m(\tilde{u}) \rfloor Q \) for all \( \lfloor m(\tilde{u}) \rfloor \).

We denote the largest strong \( r1 \)-bisimulation as \( \cong_{r1} \), and the largest weak \( r1 \)-bisimulation as \( \equiv_{r1} \).

The \( \kappa \)-versions, strong and weak **\( \kappa_1 \)-bisimulation** \( \cong_{\kappa_1} \) and \( \equiv_{\kappa_1} \), are defined by replacing \( \tau \)-bisimulation with its \( \kappa \)-version, the \( \kappa \)-bisimulation, in the above definition.

**Lemma 3-19:** The responsive equivalence and \( r1 \)-bisimulation are coincide for both \( \kappa \)-version and non-\( \kappa \)-version, i.e., \( \cong_{\kappa r1} = \equiv_{\kappa r1} \cong_{r1} = \equiv_{r1} \) and \( \cong_{r1} = \equiv_{r1} \).

**Proof:** By induction. We only show \( \cong_{r1} = \equiv_{r1} \) here, and all other cases can be proven in similar way.

\( \cong_{r1} \subseteq \equiv_{r1} \): Let \( P \cong_{r1} Q \), then we can write \( \mathcal{T}[P] \cong_{r1} \mathcal{T}[Q] \) where \( \mathcal{T}[\cdot] \).

Assume \( \mathcal{T}[P] \cong_{r1} \mathcal{T}[Q] \) is held for some responsive testing context \( \mathcal{T} \).

By the definition of \( \cong_{r1} \), for all \( \lfloor m(\tilde{u}) \rfloor \), we have \( \mathcal{T}[P] \cong_{r1} \mathcal{T}[Q] \) for each of \( \mathcal{T}[\cdot] \).

By induction, and notice \( \cong_{r1} \subseteq \cong_{r1} \) from the definition of \( \cong_{r1} \), we conclude that,

\( P \cong_{r1} Q \) implies \( \forall \mathcal{T}(\mathcal{T}[P] \cong_{r1} \mathcal{T}[Q]) \), that is, \( P \equiv_{r1} Q \), by the definition of \( \equiv_{r1} \).

\( \equiv_{r1} \subseteq \cong_{r1} \): Let \( P \equiv_{r1} Q \), then it implies \( P \equiv_{r1} \) because \( \mathcal{T}[P] \equiv_{r1} \mathcal{T}[Q] \) for \( \mathcal{T}[\cdot] \), that is, \( \equiv_{r1} \subseteq \equiv_{r1} \).

It also implies \( \lfloor m(\tilde{u}) \rfloor P \equiv_{r1} \lfloor m(\tilde{u}) \rfloor Q \) for all \( \lfloor m(\tilde{u}) \rfloor \) because \( \mathcal{T}[P] \equiv_{r1} \mathcal{T}[Q] \) for each \( \mathcal{T}[\cdot] \).

And still, it implies for each responsive testing context \( \mathcal{T} \), if we write \( \mathcal{T}[P] \equiv_{r1} \mathcal{T}[Q] \), then \( \mathcal{T}[P] \equiv_{r1} \mathcal{T}[Q] \).

In other words, \( \forall \mathcal{T}(\mathcal{T}[P] \equiv_{r1} \mathcal{T}[Q]) \). That is, \( P \equiv_{r1} Q \) implies \( \lfloor m(\tilde{u}) \rfloor P \equiv_{r1} \lfloor m(\tilde{u}) \rfloor Q \) for all \( \lfloor m(\tilde{u}) \rfloor \). Therefore \( \equiv_{r1} \subseteq \equiv_{r1} \) by the definition of \( \equiv_{r1} \).
It is easy to verify that $O_1 \approx_{\text{r}} O_2$ and $O_1 \approx_{\text{r}} O_2$ hold for the processes $O_1$ and $O_2$ mentioned in the example at earlier of this session. The r1-bisimulation provides a test platform for measuring 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) or-bisimulation $S$ such that whenever $PSQ$ then $PK(S)$ implies either $Q(P)S$ and $PK(Q)$, or $PKQ$ if $PKS$. The weak responsive bisimulation is obtained by replacing transitions with weak transitions everywhere. We denote $\sim_r$ and $\approx_r$ as the largest strong and weak responsive bisimulation respectively. Clearly, $\sim_r \subseteq \approx_r$.

The $\kappa$-versions, strong and weak $\kappa$-bisimulation $\sim_{\kappa}$ and $\approx_{\kappa}$, are defined by replace or-bisimulation with $\kappa$-bisimulation in the above definitions. Clearly, $\sim_{\kappa} \subseteq \approx_{\kappa}$.

**Lemma 3-21**: The responsive bisimulation and r1-bisimulation are coincide for both $\kappa$-version and non-$\kappa$-version, i.e., $\sim_{\kappa} \subseteq \approx_{\kappa} = \approx_{\kappa} \subseteq \approx_1 = \approx_{\kappa}$ and $\approx_{\kappa} \subseteq \approx_1$.

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

$\approx_{\kappa} \subseteq \approx_{\kappa}$: Let $R \subseteq \{(m \circ \tilde{m})P, (m \circ \tilde{m})Q), P \approx R Q \}$ so $S = \sim_{\kappa}$. Assume $[m \circ \tilde{m}]P \approx R P'$ for some an arbitrary action $\alpha$ by the rules in Figure 3-2 and by Corollary 3-16 (4), it is only possible in the following two cases:

1. $P \approx R P'$ and $\alpha \circ \tilde{m}(m \circ \tilde{m})$, then $P \approx [m \circ \tilde{m}]P'$. Since $P \approx R Q$, we have $\alpha \in \tilde{m}(m \circ \tilde{m})Q$, $P \approx R P'$ and $[m \circ \tilde{m}]Q \subseteq \approx_{\kappa} R Q$. Thus, $P \approx R P'$ is possible if $[m \circ \tilde{m}]Q \subseteq \approx_{\kappa} R Q$.

2. $\alpha \in \tau$ and $\alpha \in \tilde{m}(m \circ \tilde{m})P'$, then $P \approx R P'$ it implies $\alpha \in \tilde{m}(m \circ \tilde{m})Q$ and $P \approx R P'$ that is $(P', Q') \subseteq R$ since $R \subseteq \approx_{\kappa}$.

Then by definition of $\approx_{\kappa}$, we have $R \subseteq \approx_{\kappa}$, that is, $P \approx R P'$ implies $[m \circ \tilde{m}]P \approx R [m \circ \tilde{m}]Q$. Because $\approx_{\kappa} \subseteq \approx_{\kappa}$ and because $[m \circ \tilde{m}]$ is arbitrary here, we have $\approx_{\kappa} \subseteq \approx_{\kappa}$ by the definition of $\approx_{\kappa}$.

Corollary 3-22: The responsive bisimulation and self-similar equivalence are coincide for both $\kappa$-version and non-$\kappa$-version, i.e., $\sim_{\kappa} \equiv \approx_{\kappa} = \approx_{\kappa} \equiv \approx_{\kappa} \equiv \approx_1 = \approx_{\kappa}$.
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 congruence for autonomous processes.

**Corollary 4-23**: The responsive bisimulations are preserved by localisation. That is, let \( S \) be any of \( \sim_r, \sim_{pr}, \sim_{pr} \) or \( \sim_{pr} \), then \( PSO \) implies \( [m(b)]P S[m(b)]Q \) for all \( [m(b)] \).

**Proof**: Let \( R \) be the corresponding r-bisimulation \( (\sim_{pl}, \sim_{pl}, \sim_{pl}, \sim_{pl}) \) respect to \( S \), then by Lemma 3-21, \( PSO \) implies \( PRQ \), which then implies that \( [m(b)]P R[m(b)]Q \) for all \( [m(b)] \) according to the definition of r-bisimulation, then again by Lemma 3-21, we have \( [m(b)]P S[m(b)]Q \).

\[ \Box \]

**Lemma 4-24**: The responsive bisimulations are equivalences.

**Proof**: Here we only give the proof for the strong k-version \( \sim_{pr} \) and all other cases can be proven similarly.

- Reflexive : \( P \sim_{pr} P \) for any \( P \), according to the definition of \( \sim_{pr} \).
- Symmetric: if \( P \sim_{pr} Q \) then \( Q \sim_{pr} P \), by the definition of \( \sim_{pr} \).
- Transitive: Let \( P_1 \sim_{pr} P_2 \) and \( P_2 \sim_{pr} P_3 \), where \( \sim_{pr} \leq \sim_{pr} \) and \( \sim_{pr} \subseteq \sim_{pr} \) and therefore \( P_1 \sim_{pr} P_3 \). For arbitrary action \( \alpha \), if \( \alpha \) is not an input communication act, then \( P_1 \omega \sim_{pr} P_1 \) implies \( P_2 \omega \sim_{pr} P_2 \) and \( P_2 \sim_{pr} P_2 \), which further implies \( P_3 \alpha \sim_{pr} P_3 \) and \( P_3 \omega \sim_{pr} P_3 \), that is, \( P_3 \sim_{pr} P_3 \).

If \( \alpha \) is an input communication act, say \( \alpha = n(b) \), and \( P_1 \sim_{pr} P_1 \), then we may have either \( P_2 \sim_{pr} P_1 \) and \( P_3 \sim_{pr} P_3 \), or \( P_2 \sim_{pr} P_1 \) and \( P_3 \sim_{pr} P_3 \), or \( P_2 \sim_{pr} P_1 \) and \( P_3 \sim_{pr} P_3 \). By \( \sim_{pr} \subseteq \sim_{pr} \), we have \( \sim_{pr} \subseteq \sim_{pr} \), so \( [m(b)]P_1 \omega \sim_{pr} [m(b)]P_1 \omega \) and \( [m(b)]P_3 \omega \sim_{pr} [m(b)]P_3 \omega \). By the definition, \( [m(b)]P_1 \omega \sim_{pr} [m(b)]P_3 \omega \).

\[ \Box \]

There is a problem: the responsive bisimulations are not preserved by parallel composition in general. For instance, with the \( O_1 \) and \( O_2 \) of the previous example, we have \( O_1 \sim_{pr} O_2 \), but \( (O_1 \parallel O_2) \sim_{pr} (O_1 \parallel O_2) \) for \( O_1 \subseteq \parallel \sim_{pr} \), because the occurrence of input \( m \) in \( O_3 \) has changed the ability of \( O_1 \) on receiving message from \( m \). 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 \( m \) be the input polar of a communication channel name \( m \), \( P \) be a process for which \( m \in \text{in}(P) \), and \( S \) be the context \( S_{\{\}} \equiv (n(b)) \text{in}(Env) \{[1,1]\} \) where \( m \in \text{in}(Env) \) while \( m \) may or may not be a member of \( n \). We say that \( P \) is an owner of \( m \) (or say, \( m \) is owned by \( P \)) with respect to the environment \( Env \);

- \( Env \) is an environment free of \( m \) (or say, \( m \)-free environment);
- \( S_{\{\}} \) is an \( m \)-safe environment context, or \( m \)-safe environment for short.

An \( m \)-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 reachable states with respect to the environment $Env$, i.e., $\text{fin}(P) \cap \text{fin}(Env) = \emptyset$. 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 $\text{fin}(P) = \emptyset$.

Lemma 4-27: The process safety is preserved by evolution. That is, if $\text{fin}(P) \cap \text{fin}(Env) = \emptyset$ holds for processes $P$ and $Env$, then $\text{fin}(P') \cap \text{fin}(Env') = \emptyset$ 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 $\kappa$-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 $\star\text{m}_1(a_1, a_2)$ may be modelled as $\nu \text{msert}(\alpha)\text{msert}(\text{m}_1(a_1, a_2))$, and on the object side the encoding will look like $(\nu \text{m}(\alpha)\text{msert}(\text{m}_1(a_1, a_2)))\text{Act}(\text{m}(\alpha)\text{msert}(\text{m}_1(a_1, a_2)))$.

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

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

Let $\equivP$ be the congruence induced by the commutativity and associativity laws for parallel composition “$||$” in Figure 2-2 and rule ISum SUM2 in Figure 3-2, and let relation $\mathcal{R} \equiv \{(P_1 || P_2) | (P_1 || P_2) = (P_1 || P_2) \} \cup \kappaP$. Let $Q_1 \equiv P_1 || P_2$ and $Q_2 \equiv P_1 || P_2$, and $Q_1 \equiv P_1 || P_2$, for some action $\alpha$, by $P \equiv P_2$, it must be in one of the following three cases:

1. $P_1 \equiv P_1', P_2 \equiv P_2', P_1 \equiv P_2', P_2 \equiv P_2'$, and therefore $Q_1 \equiv P_1 || P_2$, for $Q_1 \equiv P_1 || P_2$;
2. $P_1 \equiv P_1', P_2 \equiv P_2'$, and therefore $Q_1 \equiv P_1 || P_2'$, for $Q_1 \equiv P_1 || P_2'$;
3. $\alpha \equiv \text{msert}(\alpha)$, $P_1 \equiv P_1', P_2 \equiv P_2'$, and $P_1 \equiv P_1', P_2 \equiv P_2'$ by the autonomous condition $\text{fin}(P) \cap (\text{fin}(P_1) \cup \text{fin}(P_2)) = \emptyset$.

The cases with synchronisation, such as $P_1 \equiv P_1', P_2 \equiv P_2'$, or $P_1 \equiv P_1', P_2 \equiv P_2'$, or $P_1 \equiv P_1', P_2 \equiv P_2'$, or $P_1 \equiv P_1', P_2 \equiv P_2'$, or $P_1 \equiv P_1', P_2 \equiv P_2'$, have been covered by these three above cases, according to Remark 2-5, and therefore need not be considered separately.

Since $(Q_1, Q_2) \in \mathcal{R}$ for cases 1-2, and $(Q_1, [\text{m}(\alpha)]\mathcal{Q}) \in \mathcal{R}$ for the third case, therefore $\mathcal{R}$ is a $\kappaP$ upto $\equivP$.

Proposition 4-30: The responsive bisimulations are preserved by output polarity name substitution. That is, to each of the $\kappa$-version or non-$\kappa$-version responsive bisimulations $\mathcal{S}$, $\text{PSQ}$ implies $\text{PQR}$ for all $\sigma = [\text{m}(\alpha)]\mathcal{Q}$. The cases of synchronisation, such as $P_1 \equiv P_1', P_2 \equiv P_2'$, or $P_1 \equiv P_1', P_2 \equiv P_2'$, or $P_1 \equiv P_1', P_2 \equiv P_2'$, or $P_1 \equiv P_1', P_2 \equiv P_2'$, have been covered by these three above cases, according to Remark 2-5, and therefore need not be considered separately.

Since $(Q_1, Q_2) \in \mathcal{R}$ for cases 1-2, and $(Q_1, [\text{m}(\alpha)]\mathcal{Q}) \in \mathcal{R}$ for the third case, therefore $\mathcal{R}$ is a $\kappaP$ for all $\sigma = [\text{m}(\alpha)]\mathcal{Q}$.
some \( r, \tilde{r} \) and \( P' \) such that \( m=\tilde{r} \sigma, \tilde{u}=\tilde{v} \sigma, P'=\tilde{P}_r \) and \( P=(\forall \tilde{v}(\forall \tilde{v} \sigma)) P' \). Clearly, \( P((\forall \tilde{v}(\forall \tilde{v} \sigma)) P') \), it implies \( Q((\forall \tilde{v}(\forall \tilde{v} \sigma)) Q') \) and \( P'-\sigma Q' \) since \( P'-\sigma Q \), this further implies there exists some \( Q' \) s.t. \( Q=Q((\forall \tilde{v}(\forall \tilde{v} \sigma)) Q') \). Therefore, \( Q=Q((\forall \tilde{v}(\forall \tilde{v} \sigma)) Q') \) and \( Q=Q((\forall \tilde{v}(\forall \tilde{v} \sigma)) Q') \). This matches \( (P_r, Q_r) \in \mathcal{R} \) as required.

Since \( P_{\sim \sigma} Q \), we have that, \( P_{\sim \sigma} Q' \) implies

\[
Q_{\sim \sigma} Q' \quad \text{and} \quad P_{\sim \sigma} Q'.
\]

It is only possible when \( Q=((\forall \tilde{v}(\forall \tilde{v} \sigma)) Q') \), where \( m \in \tilde{v} \sigma, A_{\sim \sigma} Q', Q'=Q_{\sim \sigma} Q' \). It is easy to see, \( Q_{\sim \sigma} Q' \) and \( (P_r, Q_r) \in \mathcal{R} \).

by transition rules, it is only possible when \( P=\tilde{P}_r \), since output polar substitution does not affect releasing keys, therefore there must exist some \( P' \) such that \( P'=\tilde{P}_r \) and \( P=\tilde{P}_r \). This implies \( P_{\sim \sigma} Q \). Since \( P_{\sim \sigma} Q \), we have \( Q_{\sim \sigma} Q' \) and \( P_{\sim \sigma} Q' \). Again by transition rules, it is only possible when \( P=\tilde{P}_r \), that is \( Q_{\sim \sigma} Q' \). Therefore, \( Q_{\sim \sigma} Q' \), and we have \( (P_r, Q_r) \in \mathcal{R} \) as required.

Since \( \sigma \) only affects to free output communication polars, this transition is only possible when \( P=\tilde{P}_r \). By the transition rules, the process \( P \) must be in the form \( P=((\forall \tilde{v}(\forall \tilde{v} \sigma)) P') \), where \( \tilde{v}=\tilde{v} \sigma, A_{\sim \sigma} Q', Q'=Q_{\sim \sigma} Q' \). It is easy to see, \( Q_{\sim \sigma} Q' \) and \( (P_r, Q_r) \in \mathcal{R} \).

by transition rules, it is only possible when \( P=\tilde{P}_r \), therefore \( Q_{\sim \sigma} Q' \), and \( (P_r, Q_r) \in \mathcal{R} \).

Put all these cases together, we have that, \( \mathcal{R} \) is a \( \sim_{\sigma} \) up to \( \equiv_{\sigma} \).

Proofs for other properties of the responsive bisimulations also have the similar difference with those for standard bisimulations or conventional \( \sigma \)-calculi, and we have to provide them all instantaneously.

**Proposition 4–31:** The responsive bisimulations are preserved by restriction. That is, to each of the \( \sigma \)-version or non-\( \sigma \)-version responsive bisimulations \( S \), whenever \( P \in S \) implies \( ((\forall \tilde{v})(\forall \tilde{v} \sigma)) P \). For these can be proven by showing \( \mathcal{R} \equiv ((\forall \tilde{v}) P, (\forall \tilde{v} \sigma) Q, P \equiv_{\sigma} Q) \cup S \) is a \( \equiv_{\sigma} \) up to \( \equiv_{\sigma} \), where \( \equiv_{\sigma} \) be the structural congruences in Figure 2–3 and Figure 3–2. Again we only show that for strong \( \sigma \)-version \( S \equiv_{\sim \sigma \sigma} \) and all other cases can be proven similarly. Let's exam all the possible transitions that \( (\forall \tilde{v}) P \) may take:

\((\forall \tilde{v}) P \equiv (\forall \tilde{v} \sigma) P \). From the structural congruence rules and transition rules, this transition is only possible when \( m \in \tilde{v} \sigma, \tilde{v} \in S \) and there exists some \( P' \) such that \( P=(\forall \tilde{v}(\forall \tilde{v} \sigma)) P' \).
where $\bar{r} = \bar{s}$. By rule tr-OUT, tr-RES and tr-PARL, we have $P^t\zeromap{m}(n_1)P'$ and $P^\approx(v_2)P'$. This implies $Q^\zeromap{m}(n_1)Q'$ and $P^\approx_{\sim_0}Q'$ since $P^\approx_{\sim_0}Q'$ by rule tr-OUT and tr-RES we have $(v_1)Q^m(n_1)(v_1)Q'$. It matches $((v_1)P', (v_1)Q')QC$ as required.

$(v_1)P^m(n_1)P'$: it is only possible when $m = \bar{a}$. Let $P^m(n_1)P'$ by rule tr-RES, $(v_1)P^m(n_1)(v_1)P'$, that is, $P^\approx(v_1)P'$. From $P^\approx_{\sim_0}Q'$, the transition $P^m(n_1)P'$ implies either $Q^m(n_1)Q'$ and $P^\approx_{\sim_0}(v_1)Q$, leading to $((v_1)P', (v_1)Q')QC$ as required, or $Q^m(n_1)Q'$ and $P^\approx_{\sim_0}(v_1)Q$. Then $(v_1)QQ_2(v_1)Q'$ by rule tr-RES, and it also matches $((v_1)P', (v_1)(m(n_1))Q)QC$ as required.

$(v_1)P^k_2P'$: it is only possible when $k = \bar{a}$ and $P^k_2P'$, that implies $Q^k_2Q'$ and $P^\approx_{\sim_0}Q'$ since $P^\approx_{\sim_0}Q'$. From rule tr-RES, $P^\approx(v_1)P'$ and $(v_1)Q^k_2(v_1)Q'$, and therefore $((v_1)P', (v_1)Q')QC$ as required.

$(v_1)P^k_2P'$: by rule tr-RES, tr-RELS and str-SCP3, it is only possible when $k = \bar{a}$ and $P^k_2P'$, that implies $Q^k_2Q'$ and $P^\approx_{\sim_0}Q'$ since $P^\approx_{\sim_0}Q'$. From rule tr-RES we have $P^\approx(v_1)P'$ and $(v_1)Q^k_2(v_1)Q'$, and therefore $((v_1)P', (v_1)Q')QC$ as required.

$(v_1)P^k_2P'$: it is only possible when $P^k_2P'$, then we have $Q^l_2Q'$ and $P^\approx_{\sim_0}Q'$ since $P^\approx_{\sim_0}Q'$. From rule tr-RES we have $P^\approx(v_1)P'$ and $(v_1)Q^l_2(v_1)Q'$, and therefore $((v_1)P', (v_1)Q')QC$ as required.

Put them together, by the definition, $Q$ is a $\approx_{\sim_0}$ up to $\equiv$.

The following proposition is equivalent to say, in the term of ordinary $\pi$-calculi, the responsive bisimulations are preserved by input prefix, replication, choice and, outside the $\pi$-calculus 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 $\kappa$-version or non-$\kappa$-version responsive bisimulations $S$, if $P_1$ and $P_2$ are autonomous processes, then $P_1SP_2$ implies $S[P_1]SP_2[P_2]$ for all process contexts $P_1$ of the form $P_1 \equiv \lambda (v_1)m(n_1)\Sigma(G)$.

**Proof:** In order to prove the proposition, we extend the $\approx_{\kappa}$ up to a more generic $\approx_{\kappa}$.

Let $R$ be an arbitrary process, $R_1$ and $R_2$ be arbitrary safe processes (this means $\mathcal{J}(R_1) \equiv \mathcal{J}(R_2)$) satisfying $R_1SR_2$ and $\bar{k}_1$ be set of keys, if we can prove that, “$P_1SP_2$ implies $Q_1SP_2$”, then $Q_1SP_2$ implies $Q_1SP_2$ for all such $P_1, R_1, R_2$, and $\bar{k}_1$”, this proposition can be concluded by letting $R = \emptyset, R_1 = \emptyset, R_2 = \emptyset$ and $\bar{k}_1 = \emptyset$. Let $\mathcal{J}(P, m(n_1)G)$ be a function that truth value of $\mathcal{J}(P, m(n_1)G)$ is $\emptyset$, let $\bar{a}$ be the set of all autonomous processes, then we show that

$$
\mathcal{R} = \{ (v_1\bar{k}_1)(A(P_1)) \mid R \mid R_1) \in (v_1\bar{k}_1)(A(P_2)) \mid R \mid R_2) : (P_1, P_2) \in \mathcal{R}, (P_1SP_2) \cap \mathcal{J}(R_1m(n_1)G), (P_2) \cap \mathcal{J}(R_2m(n_1)G)\}
$$

is a $S$. To save our writing, we define the GEC context $A[\bar{a}](v_1)m(n_1)\Sigma(G)$, that is, $A[\bar{a}] = A[\bar{a}]$, and write

$$
Q_1 \equiv (v_1\bar{k}_1)(A(P_1)) \mid R \mid R_1) \in (v_1\bar{k}_1)(A(P_2)) \mid R \mid R_2)
$$

and

$$
Q_2 \equiv (v_1\bar{k}_1)(A(P_2)) \mid R \mid R_2)
$$

and then show “whenever $P_1SP_2$ implies $Q_1SP_2$” by induction over all possible actions. Here we only show that for $S = \approx_{\bar{a}}$, all others can be proven similarly.

Whenever $Q_1SP_2$ for some action $a$, then it must satisfy $\mathcal{J}(a) \cap \bar{k}_1 = \emptyset$, and is only possible in one of the following four cases:

1. $R_1SR_2$, then $Q_1 \equiv (v_1\bar{k}_1)(A(P_1)) \mid R \mid R_1)$ and $Q_2 \equiv (v_1\bar{k}_1)(A(P_2)) \mid R \mid R_2)$, and $(Q_1, Q_2) \in \mathcal{R}$

2. $R_1R_2$, then $Q_1 \equiv (v_1\bar{k}_1)(A(P_1)) \mid R \mid R_1)$ if $a$ is not a communication input action, then $R_1R_2$ and $R_2SR_2$ since $R_1SR_2$. Then $Q_1 \equiv (v_1\bar{k}_1)(A(P_1)) \mid R \mid R_2)$ and $(Q_1, Q_2) \in \mathcal{R}$

3. $A(P_1) \mid a$, let exam all the possible action $$: 

$$
\text{must be } Q_1 \equiv (v_1\bar{k}_1)(A(P_1)) \mid R \mid R_1), \text{ and } Q_2 \equiv (v_1\bar{k}_1)(A(P_2)) \mid R \mid R_2), \text{ therefore } (Q_1, Q_2) \in \mathcal{R}
$$
\(\alpha\equiv m(\bar{u}, \bar{v})\), it must be \(A\equiv \lambda \bar{u}.A^2\) and \(Q_1\equiv (\forall \bar{u}, \bar{v}.(A^2(\bar{u}, \bar{v}) \mid R) \mid R)\) and \(\alpha = \text{add}(L, A)\), then \(Q_2\) can also commit the same action such that \(Q_2, (L, A)\) and \(Q_2\equiv (\forall \bar{u}, \bar{v}.(A^2(\bar{u}, \bar{v}) \mid R) \mid R)\). Since \(P_1\) and \(P_2\) are autonomous, and therefore \(P_1[\bar{u}, \bar{v}]\) and \(P_2[\bar{u}, \bar{v}]\) are, this further implies that \(R_1[P_1[\bar{u}, \bar{v}]]\) and \(R_2[P_2[\bar{u}, \bar{v}]]\) are safe. That is, \(m \in (R_1[P_1[\bar{u}, \bar{v}]] \circ R_2[P_2[\bar{u}, \bar{v}]]). A \equiv \bar{g}(R_1[P_1[\bar{u}, \bar{v}]]) \circ R_2[P_2[\bar{u}, \bar{v}]]\) is true. Since \(P_1, S_2, R_2\) by Proposition 4-30 and Proposition 4-29, we have \(R_1[P_1[\bar{u}, \bar{v}]] \circ R_2[P_2[\bar{u}, \bar{v}]]\), therefore \((Q_1, Q_2) \in \mathcal{R}\).

\(\alpha\equiv m(\bar{u})\) where \(m \neq m\), it must be \(A\equiv (\forall \bar{u}, \bar{v}.(A^2(\bar{u}, \bar{v}) \mid R) \mid R)\), then \(Q_1\equiv (\forall \bar{u}, \bar{v}.(A^2(\bar{u}, \bar{v}) \mid R) \mid R)\), and we can have \(Q_2\equiv Q_3\). Therefore, \((Q_1, Q_2) \in \mathcal{R}\).

4), \(\alpha\) is a transition caused by a synchronisation action between \(\mathcal{D}[P_1], R, R_1\), following the rule tr-SYNCE. This in turn can only be possible in the following sub-cases which satisfy \(\kappa \equiv \bar{k}_1\):

- \(A\equiv (\mathcal{D}[P_1]) \circ \bar{k}_1\), where \(J \equiv \{m\} \cup \text{gram}(G)\), and either \(R_1 \circ \bar{k}_1\), then \(Q_1\equiv (\forall \bar{u}, \bar{v}.(A^2(\bar{u}, \bar{v}) \mid R) \mid R)\) and \(Q_2\equiv Q_3\), where \(Q_2\equiv (\forall \bar{u}, \bar{v}.(A^2(\bar{u}, \bar{v}) \mid R) \mid R)\), therefore \((Q_1, Q_2) \in \mathcal{R}\).

- \(R_2 \circ \bar{k}_2\), then \(Q_1\equiv (\forall \bar{u}, \bar{v}.(A^2(\bar{u}, \bar{v}) \mid R) \mid R)\), and we have \(R_2 \circ \bar{k}_2\). Since \(R_1 \circ \bar{k}_1\), and \(R_2 \circ \bar{k}_2\), we have \(R_1 \circ \bar{k}_1\), therefore \((Q_1, Q_2) \in \mathcal{R}\).

Put them together, we have \(\mathcal{R} \equiv \mathcal{S}\) by the definition of \(\sim\).

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 \(\kappa\)-version or non-\(\kappa\)-version responsive bisimulation \(\mathcal{S}\), if \(P_1\) and \(P_2\) are safe processes, then \(P_1, S_2, R_2\) implies \(\mathcal{S}[P_1, S_2, P_2]\) for all process context of the form \(\mathcal{D}[\mathcal{A}] \equiv M(\bar{x}) \circ (\forall \bar{u}, \bar{v}.(A^2(\bar{u}, \bar{v}) \mid R) \mid R)\) where \(m \in L\).

**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 \(\kappa\)-version or non-\(\kappa\)-version responsive bisimulations \(\mathcal{S}\), if \(P_1\) and \(P_2\) are autonomous processes, then \(P_1, S_2, R_2\) implies \(\mathcal{S}[P_1, S_2, P_2]\) for all process context \(\mathcal{D}[\mathcal{A}]\).

**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 the bisimulations in standard \(\pi\)-calculus, may apply to responsive bisimulation.

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

- The (strong) **early responsive bisimulation** is a (strong) \(\alpha\)-bisimulation \(\mathcal{S}\) if whenever \(PSQ\) then \(P \equiv (\bar{u})P'\) implies \(\forall \exists Q\) s.t. either \(\exists Q\) or \(P \equiv (\bar{u})Q\).
- The (strong) **late responsive bisimulation** is a (strong) \(\alpha\)-bisimulation \(\mathcal{S}\) if whenever \(PSQ\) then \(P \equiv (\bar{u})P'\) implies \(\exists Q\) s.t. either \(P \equiv (\bar{u})Q\) or \(P \equiv (\bar{u})Q\).
- The (strong) **open responsive bisimulation** is a symmetric relation \(\mathcal{S}\) on processes if whenever \(PSQ\) then for any output communication substitution \(=\bar{g}(\bar{u})\), we have \(P \equiv (\bar{u})P\) implies \(\exists Q\) s.t. \(Q \equiv LQ\) and \(P \equiv (\bar{u})Q\), where either \(\alpha = (\forall \bar{u}(\bar{v})\equiv (\bar{u})\alpha\).
\( \text{Psim}(\alpha) \) implies \( \exists Q \) s.t. either \( Q_{\text{mem}(\vec{n})} \) and \( P_{\text{Sim}} \) or \( Q_{\text{Sim}} \) and \( P_{\text{Sim}(\vec{n})} \).

such that whenever \( P_{\text{Sim}} \) then \( P_{\text{Sim}(\vec{n})} \) implies \( Q_{\text{Sim}(\vec{n})} \) and \( P_{\text{Sim}} \) for all action \( \alpha \) in the form of either \( \alpha = (\forall m)(\exists n) \) or \( \alpha = \tau \), and \( \text{Sim}(\vec{n}) \cap \text{Sim}(\vec{n}) = \emptyset \).

For early and late responsive bisimulations, the \( \kappa \)-versions are defined by replace or-bisimulation with \( \kappa \)-bisimulation in the above definitions. For open responsive bisimulations, the \( \kappa \)-versions are defined by including \( \exists (\exists \vec{n}) \) into the range of \( \alpha \).

However, as the \( \kappa \)-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-\( \kappa \)-versions, let \( S \) be strong (or weak) responsive bisimulation, \( S \) be any of strong (or weak, respectively) early, late and open responsive bisimulations, then we get

\( P_{\text{Sim}} \) implies \( P_{\text{Sim}(\vec{n})} \), by letting \( \vec{n} = \vec{n} \); and \( P_{\text{Sim}(\vec{n})} \) implies \( P_{\text{Sim}} \), by applying Proposition 4-30.

Similarly we can prove it for the \( \kappa \)-versions.

**5.2 Relation with some conventional bisimulations**

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

**Definition 5-38**: The (strong) **ground bisimulation** is a (strong) or-bisimulation \( S \) if whenever \( P_{\text{Sim}} \) then \( P_{\text{Sim}(\vec{n})} \) implies either \( Q_{\text{Sim}} \).

The (strong) **early bisimulation** is a (strong) or-bisimulation \( S \) if whenever \( P_{\text{Sim}} \) then \( P_{\text{Sim}(\vec{n})} \) implies \( \forall \vec{n} \exists Q \) s.t. \( Q_{\text{Sim}} \) and \( P_{\text{Sim}(\vec{n})} \).

The (strong) **late bisimulation** is a (strong) or-bisimulation \( S \) if whenever \( P_{\text{Sim}} \) then \( P_{\text{Sim}(\vec{n})} \) implies \( \exists \vec{n} \) s.t. \( Q_{\text{Sim}(\vec{n})} \) and \( P_{\text{Sim}(\vec{n})} \).

The (strong) **open bisimulation** is a (strong) or-bisimulation \( S \) if whenever \( P_{\text{Sim}} \) then for any output name substitution \( \sigma = (\exists \vec{n}) \), \( P_{\text{Sim}(\vec{n})} \) implies \( \exists \vec{n} \) s.t. \( Q_{\text{Sim}(\vec{n})} \) and \( P_{\text{Sim}(\vec{n})} \).

For each of them the weak version is obtained by replacing transitions with weak transitions everywhere, and the \( \kappa \)-version is defined by replace or-bisimulation with \( \kappa \)-bisimulation in the above definitions. We denote \( \sim_{\text{ag}} \) (\( \sim_{\text{w}} \)) and \( \sim_{\text{g}} \) (\( \sim_{\text{w}} \)) be the largest strong (weak) \( \kappa \)-version and non-\( \kappa \)-version ground bisimulation respectively.

**Lemma 5-39**: The ground bisimulation, early bisimulation, late bisimulation and open bisimulation are all coincided in the \( \kappa \)-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, \( \sim_{\text{g}} \leq \sim_{\text{r}} \) and \( \sim_{\text{g}} \leq \sim_{\text{w}} \).

**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 \( \kappa \)-calculus:
Definition 5-41: The (strong) asynchronous bisimulation is a (strong) or-bisimulation $S$ if whenever $PSQ$ then $Pm(m')$ implies either $Qm(m')$ or $P \sim_m Q$, or $Qm(m)$, and $P' S(m(m)) Q')$.

Again, the weak asynchronous bisimulation is obtained by replacing transitions with weak transitions everywhere, and the $\kappa$-version is defined by replace or-bisimulation with $\kappa$-or-bisimulation. We denote $\sim_{\kappa}$ and $\sim_{\kappa} \approx$ be the largest strong (weak) $\kappa$-version and non-$\kappa$-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 objects. In other words, the responsive bisimulation is quite natural to compositional objects.

5.3 Relation between $\kappa$-version and non-$\kappa$-version bisimulations

For each bisimulations we have studied in the $\kappa$-calculus, its $\kappa$-version is a subset of its non-$\kappa$-version, according to their definitions. Generally, when modelling objects, the scope of a lock key $\kappa$ should not cross object boundary, and therefore the $\kappa$ and $\kappa$ 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, coordination, 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 $\kappa$-version bisimulations guarantee the similarity of coordination mechanism, and therefore the replaceability of object components. In contrary, non-$\kappa$-version bisimulations confirm the similarity of overall behaviour between objects, without knowing the details of the coordination mechanisms. When measurement of the behaviour of objects or object groups is restricted to external view, then the $\kappa$-version and non-$\kappa$-version of a bisimulation will coincide.

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

5.4 The relation between the responsive bisimulation in the polar $\pi$-calculus and in the $\kappa$-calculus

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

As we have already pointed at the beginning, the polar $\pi$-calculus, which is a sub-calculus of the $\kappa$-calculus, is not an idea tool for modelling compositional objects. From object modelling point of view, the responsive bisimulation in the polar $\pi$-calculus actually overlaps with both the $\kappa$-version and non-$\kappa$-version of that in the $\kappa$-calculus. However, the definition of responsive bisimulation in the polar $\pi$-calculus has no difference with the non-$\kappa$-version responsive bisimulation in the $\kappa$-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 \( O_1 \approx_{r} O_2 \).

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

\[
O_c \equiv (\eta)(\lambda m. (C(m, \eta) \mid F(\eta))),
\]

where \( F \equiv (\eta) (\lambda \ell . (C(m, \eta) \mid (S_\ell \{m\} \mid (\eta)))) \), \( C \equiv (\eta)(\lambda i . (C_\ell \{m\} \mid (S_\ell \{m\} \mid (\eta)))) \), \( E \equiv (\eta)(\lambda \ell . (C_\ell \{m\} \mid (S_\ell \{m\} \mid (\eta)))) \), \( S_\ell \equiv (\eta)(\lambda \ell . (C_\ell \{m\} \mid (S_\ell \{m\} \mid (\eta)))) \). \( F \) presents the functionality of \( O_c \) in the form of an object without any constraint. \( C \) presents the concurrency constraints and consists of the exclusion control \( E \) and the schedulers \( S \).

An example of scheduler can be \( S_\ell \equiv (\eta)(\lambda \ell . (C_\ell \{m\} \mid (S_\ell \{m\} \mid (\eta)))) \). In this model, the scope of the lock/unlock signal \( \kappa \) is never beyond the controller \( C \). An extension to concurrent object-oriented programming languages based on this model is presented in [Zhang01]. Now we can use \( O_c \approx_{r} O_3 \) to express the behaviour equivalence between the compositional object \( O_c \) and the non-compositional object \( O_3 \). Here we are able to use \( \approx_r \) rather than \( \approx_{r,p} \) because in these models only the objects that do not have private objects.

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

**Definition 6-42:** The safe process \( P \) is an object component process with source set \( \overline{m} \), where \( \{m\} \equiv \text{in}(P) \), if \( P \mid a \) for all \( a \in \overline{m} \).

The object component process \( C \) with source set \( \overline{m} \) is a control process with socket set \( \overline{m} \) and plug set \( \overline{m} \), if \( \{m\} \equiv \text{in}(C), \{m\} \cap \{n\} = \emptyset \), and for each \( i \) where \( \eta_i \in \overline{m} \) and \( m_i \in \overline{m} \), there exist processes \( C_i, C' \) and action sequence \( t \) which satisfies \( \{m, \overline{m}\} \cap \text{in}(t) = \emptyset \), such that \( \{C_i, C'\} \in \text{in}(D) \), \( C_i \leq C' \) and \( C' \mid C' \).

We define the generic empty control process as \( E \equiv (\eta)(\lambda \ell . (C(\overline{m}, \eta) \mid F(\eta)))) \), where \( F \equiv (\eta) (\lambda \ell . (C(m, \eta) \mid (S_\ell \{m\} \mid (\eta)))) \), \( E \equiv (\eta)(\lambda \ell . (C_\ell \{m\} \mid (S_\ell \{m\} \mid (\eta)))) \), \( S_\ell \equiv (\eta)(\lambda \ell . (C_\ell \{m\} \mid (S_\ell \{m\} \mid (\eta)))) \).

Given a control process \( D \) with socket set \( \overline{S} \) and plug set \( \overline{S} \) and an object component process \( Q \) with source set \( \overline{S} \), let \( C \equiv (\overline{S}, \overline{S}) \) and \( P \equiv (\overline{S}, \overline{S}) \), we use the abbreviation \( C \triangleright P \) to denote \( (\eta)(\lambda \ell . (C(\overline{m}, \eta) \mid F(\eta)))) \) for all \( \overline{m} \) and \( \overline{m} \), where \( \{\overline{m}, \overline{m}\} \cap \text{in}(D) = \emptyset \) and \( \{\overline{m}, \overline{m}\} \cap \text{in}(P) = \emptyset \).

One of the desired properties of the composition is identity law. With ground bisimulation, [Zhang98A] has proven the identity law for right side \( C \triangleright E \approx E \), but the left side law \( E \triangleright C \approx E \) is not generally true. With the responsive bisimulation, however, identity law is held for both sides: \( E \triangleright C \approx C \triangleright C \triangleright E \), proved by [Zhang01]. 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 constraints 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 \( m_1 \) and \( m_2 \), and the control process \( C_2 \) describes and only describes the exclusion between \( m_3 \) and \( m_4 \), then \( C_1 \triangleright C_2 \) will provide both exclusion between \( m_1 \) and \( m_2 \), and that between \( m_3 \) and \( m_4 \), but no other exclusion will be accidentally added or removed.
In other words, the effect of composition on the exclusion is to union the corresponding exclusion sets within each choice branch. For different $S'_1$ and $S'_2$, 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 $\pi$ and $S_1$ to express the composed behaviour, but the underlying 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 diagram indicates 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 $\pi$, adding the constraint decrived by $C$ to the empty control $E$, using two independent controllers $C$ and $E_0$.

![Diagram](image_url)

$E \triangleright C \Rightarrow C \triangleright E \approx_r C \mid E_0$

$\langle C_1, \pi_1 \triangleright E \rangle (\pi_2 \triangleright C_2) \approx_r C_3 \approx_r C_1 \triangleright (C_2 \triangleright C_3)$

where $\pi_1 = \pi_2 = \pi_3$ and $\pi = \pi_1 \cup \pi_2$.

Figure 6-1

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

7 Conclusion

This paper has presented the responsive bisimulation and variations in the $\kappa$-calculus. For object systems, where the input name clash can be eliminated, the responsive bisimulations are preserved by parallel composition, output name substitution and choices, 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 $\pi$-calculus, the responsive bisimulation in the $\kappa$-calculus is splitted into the $\kappa$-version, the version with the internal vision of objects, and non-$\kappa$-version, the version with external vision. Maintaining a $\kappa$-version responsive bisimulation in object components level will guarantee the non-$\kappa$-version responsive bisimulation in the whole object.

References:


