A Constraint Description Calculus for Compositional Concurrent Objects

UNSW-CSE-TR-0204

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

Abstract

This paper presents the κ-calculus, a mobile-process algebra with lock as primitive. The Guarded Conditional Exclusive Choice "$\otimes$", together with a selective locking/unlocking mechanism, is used in the κ-calculus as the only combineter for input guared processes. Therefore, for input guarded terms, the standard mutually exclusive choice "+" of CCS or π-calculus, and the parallel composition "|", become two extreme cases of the unified combinerer "$\otimes$". The κ-calculus can give a simpler, clearer and more composible description of the method exclusion in the modelling of concurrent objects, while preserves other powers of the π-calculus in modelling the mobility of concurrent objects.

An concurrent object may be modelled in the κ-calculus as either a single object process or the composition of a function object proecess and a set of control object processes. A single object process modelled in the κ-calculus has a generic form $\Lambda\circ\lbrack\mathbb{G}«\tilde{M}»\rbrack$*, where* Λ *records the statues of lock,* $\mathbb{G}$ *decribes the methods exclusion and* $\tilde{M}$ *is a set of processes each of which presents the functional behaviour of a method body.*

The κ-calculus providers a straightforward model to separate aspects such as object functionality, method exclusion and locking schema and states in a high level abstraction, and provider semantic for a compositional concurrent object-oriented programming language.

A Constraint Description Calculus for Compositional Concurrent Objects

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

1 Introduction

Programming for concurrent systems is a complicated, difficult, and problematical task, requires experience and skill. It has suffered from the lack of an easy to be mastered formal method tool for software developers to describing current behaviours of objects, reasoning about the correctness of concurrent object systems and detecting deadlock.

The π-calculus is algebra for mobile processes ([Milner92], [Milner96]), and provides a formal foundation for modelling systems with dynamic structure. It can be used to mathematically model concurrent and distributed processes, analyse their behaviour and identify deadlocks. With its ability to directly model dynamic reference structures, the π-calculus has been applied to modelling concurrent object systems ([Walker95], [Jones93], [Sangiorgi96], [Hüttel96], [Zhang97]). Some researchers ([Schneider97], [Zhang98A], [Zhang98B]) also have applied it in modelling compositional of concurrent objects in aspect-oriented programming style ([Aksit92], [Holmes97]) to avoid the inheritance anomaly [McHale94].

To motivate our work, we review the concurrent object models of [Zhang98A] and [Zhang98B] using the π-calculus (This paper presumes the readers familiar with the π-calculus). The behaviour of a concurrent object can be modelled as the parallel composition of a process F representing the object's functional behaviour with no constraint on its concurrent interactions, and a process C representing the constraints on the object's concurrent behaviour. For example, the functionality of a buffer object can be described by $F \overset{\text{def}}{=} !n_r(x).M_r\langle x\rangle \mid !n_w(x).M_w\langle x\rangle$, where $n_r(x).M_r\langle x\rangle$ and $n_w(x).M_w\langle x\rangle$ represent the behaviour of the read and write methods respectively; each of them can have unlimited invocations executing in parallel with no concern for any potential interference. To discipline those invocations, assume a synchronisation behaviour modelled by the control process $C \overset{\text{def}}{=} m_r(x).\overline{n}_r\langle x\rangle + m_w(x).\overline{n}_w\langle x\rangle$, where the choice operator in fact represents a mutual exclusion lock on those methods. Then the parallel composition of the two processes, $(\nu n)(C \mid F)$, will be weakly bisimilar to $R \overset{\text{def}}{=} m_r(x).M_r\langle x\rangle + m_w(x).M_w\langle x\rangle$, as expected.

However, the general exclution relations between object methods are difficult to be presented efficiently and compositionally in the π-calculu and most variations. For example, assume we add the third method m_3 to the above buffer object, and are given the exclusive requirements that mutually exclusive should be maintained between method m_r and m_w, and fully concurrent execution is allowed between method m_r and m_3. These two requirements may be modelled in the π-calculu respectively as the two control processes $C_1 \overset{\text{def}}{=} m_r(x).\overline{n}_r\langle x\rangle + m_w(x).\overline{n}_w\langle x\rangle$ and $C_2 \overset{\text{def}}{=} m_r(x).\overline{n}_r\langle x\rangle \mid !m_3(x).\overline{n}_3\langle x\rangle$.

Now consider the following different cases on addition requirements

1) Fully concurrent execution is also allowed between method m_w and m_3, i.e., $C_3 \overset{\text{def}}{=} m_w(x).\overline{n}_w\langle x\rangle \mid !m_3(x).\overline{n}_3\langle x\rangle$;
2) Mutually exclusive should be maintained between method m_2 and m_3, i.e., $C_3 \overset{\text{def}}{=} m_w(x).\overline{n}_w\langle x\rangle + m_3(x).\overline{n}_3\langle x\rangle$;
3) The same as 1), except the reading method m_r should not mutually exclusive with itself, but must be mutually exclusive with the writing method m_w.

For 1), it is easy to put the addition requirement together with the previous ones to construct a composed control process: $C \overset{\text{def}}{=} (\, m_r(x).\overline{n}_r\langle x\rangle + m_w(x).\overline{n}_w\langle x\rangle\,) \mid !m_3(x).\overline{n}_3\langle x\rangle$. However, it is difficult for 2) and 3), because:

a) The entire control C has to be rewritten from scratch merely for a minor change in requirement, without re-using of the existing 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. In the algebra of exclusion, the relation between m_1 and m_2 is presented as $m_1 \times m_2$ for mutually exclusive, as $m_1 \mid m_2$ for no exclusive, and the abbreviation $\overline{m}_1$ stands for self exclusive $m_1 \times m_1$. Therefore the above three situations can be described in turn as $\overline{m}_r \times \overline{m}_w \mid m_3$, $\overline{m}_r \times \overline{m}_w \mid \overline{m}_w \times \overline{m}_3$, and $m_r \times \overline{m}_w \mid \overline{m}_3$.

On another hand, the algebra of exclusion is static, unable to present the dynamical behaviour of concurrent objects, and almost always resulting in a more restricted synchronisation when modelling a composition of objects.

In this paper we propose the κ-calculus, an extended process algebra for modelling of concurrent objects, which welds the mobility power of the π-calculus with the synchronisation expressiveness of the algebra of exclusion ([Noble00]), and therefore can give simpler and clearer descriptions of method exclusion, while modelling dynamic behavour of concurrent objects. Like the polar π-calculus (π_p-calculus) of [Zhang02A], the κ-calculus distinguishes the input polar and output polar of a communication channel. In the κ-calculus, the major change to the polar π-calculus is that, the "+" operation, a mutually exclusive non-deterministic choice which eliminates all the un-chosen branches whenever a branch is chosen, is replaced by the conditional exclusive choice operator "⊗" where the un-chosen branches are only selectively and temperately blocked. To input-guarded processes, the '+' and '$\mid$' compositions can be viewed as two extrem cases of "⊗": permenately block all branches, and block none. In this sense, we may view the κ-calculus as a super-calculus of the polar π-calculus ([Zhang02A]).

With the κ-calculus, a control can be presented in the genenric form $C \overset{\text{def}}{=} (\tilde{{}^{\dagger}m},\tilde{\bar{n}}) \lfloor\,\rfloor \circ [\mathbb{G} \langle\!\langle \{S_i \langle \bar{n}_i \rangle_{i \in I}\} \rangle\!\rangle \langle \tilde{{}^{\dagger}m} \rangle]$, where $\mathbb{G}$ specifies the exclusion relations, and each S_i gives some scheduling information such as unlocking or early return on the ith method. As the example, for each of the previous mentioned three situations we may write the $\mathbb{G}$ as:

1) $!(\nu\,\kappa){}^{\dagger}m_r(\tilde{x})\check{\kappa}@\{{}^{\dagger}m_r,{}^{\dagger}m_w\}.\,\eta_r\langle\tilde{x},\hat{\kappa}\rangle \otimes\, !(\nu\,\kappa){}^{\dagger}m_w(\tilde{x})\check{\kappa}@\{{}^{\dagger}m_r,{}^{\dagger}m_w\}.\,\eta_w\langle\tilde{x},\hat{\kappa}\rangle \otimes\, !(\nu\,\kappa){}^{\dagger}m_3(\tilde{x})\check{\kappa}@\varnothing.\,\eta_3\langle\tilde{x},\hat{\kappa}\rangle$;
2) $!(\nu\,\kappa){}^{\dagger}m_r(\tilde{x})\check{\kappa}@\{{}^{\dagger}m_r,{}^{\dagger}m_w\}.\,\eta_r\langle\tilde{x},\hat{\kappa}\rangle \otimes\, !(\nu\,\kappa){}^{\dagger}m_w(\tilde{x})\check{\kappa}@\{{}^{\dagger}m_r,{}^{\dagger}m_w,{}^{\dagger}m_3\}.\,\eta_w\langle\tilde{x},\hat{\kappa}\rangle \otimes\, !(\nu\,\kappa){}^{\dagger}m_3(\tilde{x})\check{\kappa}@\{{}^{\dagger}m_r,{}^{\dagger}m_w\}.\,\eta_3\langle\tilde{x},\hat{\kappa}\rangle$;
3) $!(\nu\,\kappa){}^{\dagger}m_r(\tilde{x})\check{\kappa}@\{{}^{\dagger}m_w\}.\,\eta_r\langle\tilde{x},\hat{\kappa}\rangle \otimes\, !(\nu\,\kappa){}^{\dagger}m_w(\tilde{x})\check{\kappa}@\{{}^{\dagger}m_r,{}^{\dagger}m_w\}.\,\eta_w\langle\tilde{x},\hat{\kappa}\rangle \otimes\, !(\nu\,\kappa){}^{\dagger}m_3(\tilde{x})\check{\kappa}@\varnothing.\,\eta_3\langle\tilde{x},\hat{\kappa}\rangle$;

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

Related works: Some other authors, excluded the choice operator from the premier of their version of π-calculi, especially asynchronours π-calculi ([Honda91], [Honda92], [Pierce95], [Pierce96]), and present it as a higher level term encoded from other operators [Nestmann96]. However, encoding from lower level is usually too complicate for ordinary developers to use in pratics, and sometimes involved in problems like divergence we want to avoid.

Structure of the report: The rest of this report is structured as follows: section 2 briefly introduces the κ-calculus and related notions; section 3 gives more detailed discussion on locking scheme, a component of the κ-calculus; section 4 gives the communication semantics of the κ-calculus; section 5 discusses the bisimulation equivalences for process terms in the κ-calculus; section 6 discusses equivalences for choice terms in the κ-calculus; section 7 introduces the higher-order extension to the κ-calculus; section 8 presents a simple type system for the κ-calculus; section 9 and 10 encoding the π-calculus and algebra of exclusion, respectively, into the κ-calculus; section 11 discusses some further issues involved in the κ-calculus and its application in object modelling, and concludes the paper.

2 The basic κ-calculus

The κ-calculus has some features common with the polar π-calculus of [Zhang02A]: it is an asynchronous mobility process calculus close to the asynchronous π-calculus ([Amadio96] and [Hüttel96]), that is, an output action does not block other actions; it adopts the concept similar to polarised names in [Odersky95a], that is, an communication channel name m has two polars, the input polar ${}^{\dagger}m$ and the output polar $\bar{m}$, which can be considered as the input port from, and the output port to, respectively, the channel m, and a communication can only be performed by sending message from the ouput port and received by the input port of the same name; it syntactically includes a restriction which [Ravara97] and some others used spiritually, that is, only the output polar of names can be transmitted through a communication channel. All these issues have been discussed in details in [Zhang02A].

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

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

$$P::=\mathbf{0_P}\,|\,\bar{m}\langle\tilde{u}\rangle\,|\,\hat{\kappa}\,|\,(\nu\,\tilde{n})P\,|\,P_1\,|\,P_2\,|\,\Lambda\circ[G]\,|\,A\langle\tilde{a}\rangle\,|\,\eta, \qquad G::=B\,|\,(\nu\,\tilde{n})G\,|\,G_1\otimes G_2\,|\,D\langle\tilde{a}\rangle\,|\,\mathbb{D}«\tilde{P}»$$
$$B::=\mathbf{0_G}\,|\,!\beta.P\,|\,!(\nu\,\kappa)\beta.P, \qquad \beta::=\overset{+}{m}(\tilde{x})L, \qquad L::=\check{\kappa}@J, \qquad J::=\{\tilde{\overset{+}{m}}\}\,|\,\varnothing\,|\,\boldsymbol{M}$$

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

This can be viewed as the combination of two languages, the communication calculus (with in turn can be viewed as consists of P and G two sub languages) and an independent locking state calculus (Λ language). The terms η, $\mathbb{G}«\tilde{P}»$ and $«\tilde{\eta}»G$ are related to the higher order extension, let's completely ignore them in this moment, and defer their discussion until the section 6. Most other terms in the P sub language are similar to normal π-calculi: $\mathbf{0_P}$ indicates the inactive (terminated) process; $\bar{m}\langle\tilde{u}\rangle$ is the output action which send a set of output polars $\tilde{u}$ into the channel m; $(\nu\,\tilde{n})P$ binds the set of names $\tilde{n}$ within the scope of P, both polars of each the same name in $\tilde{n}$ should present in pair; $P_1\,|\,P_2$ indicates two processes run in parallel; $A\langle\tilde{a}\rangle$ is an instance of parameterised process agent, given by $A\overset{\text{def}}{=}(\tilde{w})P$, where $(\tilde{w})P$ is a process agent abstraction, obeying $((\tilde{w})P)\langle\tilde{a}\rangle\equiv P\{{}^{\tilde{a}}/_{\tilde{w}}\}$. For the rest two terms, $\hat{\kappa}$ is the action which emits the *unlock signal* within the scope where the name κ is bound; and $\Lambda\circ[G]$ is the *guarded conditional exclusive choice* (GEC choice), where G defines the exclusion behaviour which can place some locks on the process itself, and Λ records the lock status.

For the choice terms (G–terms), B is a choice branch; $G_1\otimes G_2$ is the choice composition; $D\langle\tilde{a}\rangle$ is instance of parameterised choice agent, given by $D\overset{\text{def}}{=}(\tilde{w})G$, where $(\tilde{w})G$ is a choice agent abstraction, obeying $((\tilde{w})G)\langle\tilde{a}\rangle\equiv G\{{}^{\tilde{a}}/_{\tilde{w}}\}$; $\mathbf{0_G}$ is the unreachable choice, in the future we can omit the subscript of both $\mathbf{0_G}$ and $\mathbf{0_P}$ without any ambiguity. Unlike that in π, every branch B here always behaves as a (lazy) replication, among them, "$!(\nu\,\kappa)$" creates a fresh key κ private to each replicated copy; in $\beta.P$ the action prefix operator "." indicates the execution of action β before the execution of the continuation process P; the action $\overset{+}{m}(\tilde{x})L$, where we stipulate that no name appears in both $\tilde{x}$ and L, ie., $\{\tilde{x}\}\cap n(L)=\varnothing$, produces two simultaneous events without any time interval: receiving information $\tilde{x}$ from the input port of channel m, and triggering the lock L; the lock $L=\check{\kappa}@J$ read as "lock all input channels in J with key κ", where the exclusion set J specifies the channels to be locked within the GEC choice and κ is the key for unlocking the lock; $\boldsymbol{M}$ is the entire ${}^{+}\mathcal{M}$, the set of input polar of all channel names, and therefore $J=\boldsymbol{M}$ will enforce the locking of every channel within the GEC choice.

It worth noting that an external observer may not detect the L component of the action $\overset{+}{m}(\tilde{x})L$ but only $\overset{+}{m}(\tilde{x})$, since L is totally internal to the GEC choice while $\overset{+}{m}(\tilde{x})$ represents a commitment to an outside message. It is also worth to noting that unlike some asynchronous π-calculi, here is no τ prefix in a GEC choice.

The other part of the process $\Lambda\circ[G]$, Λ, acts as a state machine maintaining and monitoring the current status of locks, and a communication through a channel m which is recorded in Λ as being locked by some key κ, will not be able to reach G, until all the lockings on m have removed from Λ by some unlocking actions. More details of Λ are discussed in the next section.

The basic idea of transitions in this calculus can be described as the follows:

> Responses to an output action $\overline{m}\langle\tilde{u}\rangle$ occurred somewhere in the environment, if in a *GEC* choice $\Lambda\circ\lceil G\rceil$ the channal m is not blocked by Λ, then the branch $B \overset{\text{def}}{=} !{}^{\dagger}m(\tilde{x})\kappa@J.P$ may commit on the input action ${}^{\dagger}m(\tilde{u})$ and fire the lock $\kappa@J$ at the same time. Such a commitment results in the execution of a copy of process P with all the free occurancies of $\tilde{x}$ in P replaced by $\tilde{u}$. The trigger of lock $\kappa@J$ results in a state change of Λ such that any branch of G with a channal name included in J will be blocked by key κ in Λ. Note, the communication action is an one step action, should NOT be considered as a catenation of two successive steps ${}^{\dagger}m(\tilde{u})$ and $\check{\kappa}@J$. In other words, there is no intermedia status between the finish of ${}^{\dagger}m(\tilde{u})$ and start of emitting $\kappa@J$.
>
> Responses to a unlocking signal $\hat{\kappa}$ occurred somewhere in the environment, in a *GEC* choice $\Lambda\circ\lceil G\rceil$, Λ will change its status by remove a lock of key κ from itself.

Different Λ syntax and semantics will give different locking schemes and locking status evolution paths, but will not interfere with semantic or syntax of the G language, or vice versa. Based on the Λ language introduced in the next section, an example of a commitment between a pair of output-input action over the channel m can be

$$\overline{m}\langle\tilde{u}\rangle \mid \lfloor\,\rfloor\circ\lceil\, !(\nu\kappa)\,{}^{\dagger}m(\tilde{x})\,\kappa@[{}^{\dagger}m,{}^{\dagger}m_1].M \otimes G\rceil \longrightarrow (\nu\kappa)\,(M\{\tilde{u}/\tilde{x}\} \mid \lfloor\check{\kappa}@[{}^{\dagger}m,{}^{\dagger}m_1]\rfloor\circ\lceil\, !(\nu\kappa){}^{\dagger}m(\tilde{x})\kappa@[{}^{\dagger}m,{}^{\dagger}m_1].M \otimes G\rceil\,)$$

and as the comparation, the communication modelled in the standard π-calculus could be either

$$\overline{m}\langle\tilde{u}\rangle \mid (\,m(\tilde{x}).M + G) \longrightarrow M\{\tilde{u}/\tilde{x}\} \qquad \text{or} \qquad \overline{m}\langle\tilde{u}\rangle \mid !m(\tilde{x}).M \mid G \longrightarrow M\{\tilde{u}/\tilde{x}\} \mid !m(\tilde{x}).M \mid G$$

Since the trigger of lock is totally internal to a *GEC* term (and any process P which contents it), and only the ${}^{\dagger}m_i(\tilde{u})$ part of the commucation action can be obversed from outside, in an external view we say that P commits an action ${}^{\dagger}m_i(\tilde{u})$ and reduces to P', denotes as $P\xrightarrow{{}^{\dagger}m_i(\tilde{u})}P'$. Generally, given a process P, if there exists a process Q such that $P\xrightarrow{{}^{\dagger}m(\tilde{u})}Q$, we say that P can commit on ${}^{\dagger}m$ (denoted as $P{\downarrow}{}^{\dagger}m$), otherwise we say that P cannot commit on ${}^{\dagger}m$ (denoted as $P{\not\downarrow}{}^{\dagger}m$). For example, $(\lfloor\check{\kappa}@{}^{\dagger}m_i\rfloor\circ\lceil !{}^{\dagger}m_i(\tilde{x})L_i.Q_i\otimes G\rceil\,){\not\downarrow}{}^{\dagger}m_i$, and when $m_i{\neq}m_j$, $(\lfloor\,\rfloor\circ\lceil !{}^{\dagger}m_j(\tilde{x})L_j.Q_j\rceil\,){\not\downarrow}{}^{\dagger}m_i$.

For unlocking, an example based on the same Λ language used by the above example can be

$$\hat{\kappa}_i \mid \lfloor\, \check{\kappa}_i@{}^{\dagger}m_i, \check{\kappa}_i@{}^{\dagger}m_j, \check{\kappa}_k@{}^{\dagger}m_k\rfloor\circ\lceil G\rceil \longrightarrow \lfloor\check{\kappa}_k@{}^{\dagger}m_k\rfloor\circ\lceil G\rceil$$

In the unlocking example, the process term $P\overset{\text{def}}{=}\lfloor\, \check{\kappa}_i@{}^{\dagger}m_i, \check{\kappa}_i@{}^{\dagger}m_j, \check{\kappa}_k@{}^{\dagger}m_k\rfloor\circ\lceil G\rceil$ can react to the signal $\hat{\kappa}_i$ to remove the locks with key κ_i. We may say that the process P commits on an input action on the signal channel κ_i, denoted as $P{\downarrow}\check{\kappa}_i$, or say, P performs an unlocking action $\check{\kappa}_i$ and reduces to $P'\overset{\text{def}}{=}\lfloor\check{\kappa}_k@{}^{\dagger}m_k\rfloor\circ\lceil G\rceil$, denote as $P\xrightarrow{\check{\kappa}_i}P'$. $P\xrightarrow{\check{\kappa}_i}P'$.

For convenience, we introduce some abbreviations

> **Notation 2‒1**: Abbreviation $L{=}(\nu)@J$ indicates a lock with an anonymous key, which we may also present as "$L{=}\kappa@J$ with $\kappa{=}(\nu)$" for the purpose of generic. It makes the expression $!{}^{\dagger}m(\tilde{x}\,)(\nu)@J.\,P$ to be the same as $!(\nu\,\kappa){}^{\dagger}m(\tilde{x})\kappa@J.P$ when κ is not appeared freely in P. In other words, $(\nu)@J$ is an unreleasable lock.
>
> We also abbreviate $\kappa@[{}^{\dagger}m]$ as $\kappa@{}^{\dagger}m$ whenever there is no ambiguity.
>
> **Notation 2‒2**: P‒term abbreviation $\prod P_i \overset{\text{def}}{=} P_1 \mid P_2 \mid \ldots \mid P_n$, and G‒term abbreviation $\bigotimes B_i \overset{\text{def}}{=} B_1\otimes B_2\otimes\ldots\otimes B_n$.
>
> **Notation 2‒3**: Given output actions are not blocking, we may use the P‒term abbreviated $\overline{m}\langle\tilde{u}\rangle.P\overset{\text{def}}{=}(\overline{m}\langle\tilde{u}\rangle \mid P)$ and $\hat{\kappa}.P\overset{\text{def}}{=}(\hat{\kappa} \mid P)$ to reduce the number of brackets.
>
> We may also use the P‒term abbreviation ${}^{\dagger}m(\tilde{x}).P\overset{\text{def}}{=}\lfloor\,\rfloor\circ\lceil !{}^{\dagger}m(\tilde{x})(\nu)@[{}^{\dagger}m].P\rceil$ whenever there is no ambiguity.
>
> **Notation 2‒4**: We abbreviate the generic form of output actions $(\nu\,\tilde{v})\overline{m}\langle\tilde{u}\rangle$, as $\overline{m}\langle\tilde{u}\rangle$ when $\tilde{v}{=}\varnothing$; as $\overline{m}(\tilde{u})$ when $\tilde{v}=\tilde{u}$; and as $\overline{m}$ when $\tilde{u}=\varnothing$ or $\tilde{u}$ is not important.

We may simplify the G−term $!{}^{\dagger}m({}^{\dagger}\tilde{x})L.P$ as $!{}^{\dagger}m.P$ when $\tilde{x}$ and L are not of interest, and there is no ambiguity.

In order to simplify our presentation of reduction rules, we need some short hand notations (auxiliary functions):

Definition 2−5: The label of a polar is definied by $\mathit{lab}({}^{\dagger}m)\overset{\text{def}}{=}m$, $\mathit{lab}(\bar{m})\overset{\text{def}}{=}m$, $\mathit{lab}(\check{\kappa})\overset{\text{def}}{=}\kappa$ and $\mathit{lab}(\hat{\kappa})\overset{\text{def}}{=}\kappa$.
The input polar of a name is defined by $\mathit{inp}(m)\overset{\text{def}}{=}{}^{\dagger}m$ and $\mathit{inp}(\check{\kappa})\overset{\text{def}}{=}\kappa$.
The output polar of a name is defined by $\mathit{outp}(m)\overset{\text{def}}{=}\bar{m}$ and $\mathit{outp}(\hat{\kappa})\overset{\text{def}}{=}\kappa$.

Definition 2−6: As usual, we need auxiliary functions fn, bn and n to identify the sets of free, bound and all names, respectively, of a term or action. As a calculus with polars, we also need more specified functions to identify polars. For process term, we define:

$$\begin{array}{llll}
\mathit{in}(\mathbf{0_P})\overset{\text{def}}{=}\varnothing; & \mathit{bin}(\mathbf{0_P})\overset{\text{def}}{=}\varnothing; & \mathit{on}(\mathbf{0_P})\overset{\text{def}}{=}\varnothing; & \mathit{bon}(\mathbf{0_P})\overset{\text{def}}{=}\varnothing; \\
\mathit{in}(\bar{m}\langle\bar{\tilde{u}}\rangle)\overset{\text{def}}{=}\varnothing; & \mathit{bin}(\bar{m}\langle\bar{\tilde{u}}\rangle)\overset{\text{def}}{=}\varnothing; & \mathit{on}(\bar{m}\langle\bar{\tilde{u}}\rangle)=\{m,\tilde{u}\}; & \mathit{bon}(\bar{m}\langle\bar{\tilde{u}}\rangle)\overset{\text{def}}{=}\varnothing; \\
\mathit{in}(P_1|P_2)\overset{\text{def}}{=}\mathit{in}(P_1)\cup\mathit{in}(P_2); & \mathit{bin}(P_1|P_2)\overset{\text{def}}{=}\mathit{bin}(P_1)\cup\mathit{bin}(P_2); & \mathit{on}(P_1|P_2)\overset{\text{def}}{=}\mathit{on}(P_1)\cup\mathit{on}(P_2); & \mathit{bon}(P_1|P_2)\overset{\text{def}}{=}\mathit{bon}(P_1)\cup\mathit{bon}(P_2); \\
\mathit{in}(\Lambda\circ[G])\overset{\text{def}}{=}\mathit{in}(\Lambda)\cup\mathit{in}(G); & \mathit{bin}(\Lambda\circ[G])\overset{\text{def}}{=}\mathit{bin}(G); & \mathit{on}(\Lambda\circ[G])\overset{\text{def}}{=}\mathit{on}(G); & \mathit{bon}(\Lambda\circ[G])\overset{\text{def}}{=}\mathit{bon}(G); \\
\mathit{in}((\nu\,\tilde{n})R)\overset{\text{def}}{=}\{\tilde{n}\}\cup\mathit{in}(R); & \mathit{bin}((\nu\,\tilde{n})R)\overset{\text{def}}{=}\{\tilde{n}\}\cup\mathit{bin}(R); & \mathit{on}((\nu\,\tilde{n})R)\overset{\text{def}}{=}\{\tilde{n}\}\cup\mathit{on}(R); & \mathit{bon}((\nu\,\tilde{n})R)\overset{\text{def}}{=}\{\tilde{n}\}\cup\mathit{bon}(R); \\
\mathit{in}(({}^{\dagger}\tilde{u},\bar{\tilde{v}})R)\overset{\text{def}}{=}\{\tilde{u}\}\cup\mathit{in}(R); & \mathit{bin}(({}^{\dagger}\tilde{u},\bar{\tilde{v}})R)\overset{\text{def}}{=}\{\tilde{u}\}\cup\mathit{bin}(R); & \mathit{on}(({}^{\dagger}\tilde{u},\bar{\tilde{v}})R)\overset{\text{def}}{=}\{\tilde{v}\}\cup\mathit{on}(R); & \mathit{bon}(({}^{\dagger}\tilde{u},\bar{\tilde{v}})P)\overset{\text{def}}{=}\{\tilde{v}\}\cup\mathit{bon}(R); \\
\mathit{in}(\mathbf{0_G})\overset{\text{def}}{=}\varnothing; & \mathit{bin}(\mathbf{0_G})\overset{\text{def}}{=}\varnothing; & \mathit{on}(\mathbf{0_G})\overset{\text{def}}{=}\varnothing; & \mathit{bon}(\mathbf{0_G})\overset{\text{def}}{=}\varnothing; \\
\mathit{in}(!B)\overset{\text{def}}{=}\mathit{in}(B); & \mathit{bin}(!B)\overset{\text{def}}{=}\mathit{bin}(B); & \mathit{on}(!B)\overset{\text{def}}{=}\mathit{on}(B); & \mathit{bon}(!B)\overset{\text{def}}{=}\mathit{bon}(B); \\
\mathit{in}(G_1\otimes G_2)\overset{\text{def}}{=}\mathit{in}(G_1)\cup\mathit{in}(G_2); & \mathit{bin}(G_1\otimes G_2)\overset{\text{def}}{=}\mathit{bin}(G_1)\cup\mathit{bin}(G_2); & \mathit{on}(G_1\otimes G_2)\overset{\text{def}}{=}\mathit{on}(G_1)\cup\mathit{on}(G_2); & \mathit{bon}(G_1\otimes G_2)\overset{\text{def}}{=}\mathit{bon}(G_1)\cup\mathit{bon}(G_2); \\
\multicolumn{4}{l}{\mathit{in}({}^{\dagger}m_1({}^{\dagger}\tilde{x})\check{\kappa}@[{}^{\dagger}\tilde{m}].P)\overset{\text{def}}{=}\{m_1,\kappa,\tilde{m}\}\cup\mathit{in}(P);} \\
 & \mathit{bin}({}^{\dagger}m_1({}^{\dagger}\tilde{x})L.P)\overset{\text{def}}{=}\mathit{bin}(P); & \mathit{on}({}^{\dagger}m({}^{\dagger}\tilde{x})L.P)\overset{\text{def}}{=}\{\tilde{x}\}\cup\mathit{on}(P); & \mathit{bon}({}^{\dagger}m({}^{\dagger}\tilde{x})L.P)\overset{\text{def}}{=}\{\tilde{x}\}\cup\mathit{bon}(P);
\end{array}$$

For actions, we define:

$$\begin{array}{lllll}
\mathit{in}((\nu\,\tilde{v})\bar{m}\langle\bar{\tilde{u}}\rangle)\overset{\text{def}}{=}\{\tilde{v}\}; & \mathit{in}({}^{\dagger}m({}^{\dagger}\tilde{u}))\overset{\text{def}}{=}\{m\}; & \mathit{in}(\check{\kappa})\overset{\text{def}}{=}\{\kappa\}; & \mathit{in}(\hat{\kappa})\overset{\text{def}}{=}\varnothing; & \mathit{in}(\tau)\overset{\text{def}}{=}\varnothing; \\
\mathit{bin}((\nu\,\tilde{v})\bar{m}\langle\bar{\tilde{u}}\rangle)\overset{\text{def}}{=}\{\tilde{v}\}; & \mathit{bin}({}^{\dagger}m({}^{\dagger}\tilde{u}))\overset{\text{def}}{=}\varnothing; & \mathit{bin}(\check{\kappa})\overset{\text{def}}{=}\varnothing; & \mathit{bin}(\hat{\kappa})\overset{\text{def}}{=}\varnothing; & \mathit{bin}(\tau)\overset{\text{def}}{=}\varnothing; \\
\mathit{on}((\nu\,\tilde{v})\bar{m}\langle\bar{\tilde{u}}\rangle)\overset{\text{def}}{=}\{m\}\cup\{\tilde{v}\}\cup\{\tilde{u}\}; & \mathit{on}({}^{\dagger}m({}^{\dagger}\tilde{u}))\overset{\text{def}}{=}\{\tilde{u}\}; & \mathit{on}(\check{\kappa})\overset{\text{def}}{=}\varnothing; & \mathit{on}(\hat{\kappa})\overset{\text{def}}{=}\{\kappa\}; & \mathit{on}(\tau)\overset{\text{def}}{=}\varnothing; \\
\mathit{bon}((\nu\,\tilde{v})\bar{m}\langle\bar{\tilde{u}}\rangle)\overset{\text{def}}{=}\{m\}\cup(\{\tilde{v}\}\cap\{\tilde{u}\}); & \mathit{bon}({}^{\dagger}m({}^{\dagger}\tilde{u}))\overset{\text{def}}{=}\varnothing; & \mathit{bon}(\check{\kappa})\overset{\text{def}}{=}\varnothing; & \mathit{bon}(\hat{\kappa})\overset{\text{def}}{=}\varnothing; & \mathit{bon}(\tau)\overset{\text{def}}{=}\varnothing;
\end{array}$$

And for both P terms and actions, we define

$$\mathit{fin}(t)\overset{\text{def}}{=}\mathit{in}(t)-\mathit{bin}(t);\quad \mathit{fon}(t)\overset{\text{def}}{=}\mathit{on}(t)-\mathit{bon}(t);\quad \mathit{fn}(t)\overset{\text{def}}{=}\mathit{fin}(t)\cup\mathit{fon}(t);\quad \mathit{bn}(t)\overset{\text{def}}{=}\mathit{bin}(t)\cup\mathit{bon}(t);$$
$$n(t)\overset{\text{def}}{=}\mathit{in}(t)\cup\mathit{on}(t)\equiv\mathit{fn}(t)\cup\mathit{bn}(t).$$

Since in the κ-calculus we distinguish communication channel names and keys, finer grained functions are needed. For communication channels only, we define:

$$\mathit{inc}(t)\overset{\text{def}}{=}\mathit{in}(t)\cap\mathcal{M};\quad \mathit{finc}(t)\overset{\text{def}}{=}\mathit{fin}(t)\cap\mathcal{M};\quad \mathit{binc}(t)\overset{\text{def}}{=}\mathit{bin}(t)\cap\mathcal{M};\quad \mathit{fnc}(t)\overset{\text{def}}{=}\mathit{fn}(t)\cap\mathcal{M};\quad \mathit{bnc}(t)\overset{\text{def}}{=}\mathit{bn}(t)\cap\mathcal{M};\quad \mathit{nc}(t)\overset{\text{def}}{=}n(t)\cap\mathcal{M};$$
$$\mathit{onc}(t)\overset{\text{def}}{=}\mathit{on}(t)\cap\mathcal{M};\quad \mathit{fonc}(t)\overset{\text{def}}{=}\mathit{fon}(t)\cap\mathcal{M};\quad \mathit{bonc}(t)\overset{\text{def}}{=}\mathit{bon}(t)\cap\mathcal{M}.$$

And for keys only, we define:

$$\mathit{ink}(t)\overset{\text{def}}{=}\mathit{in}(t)\cap\mathcal{K};\quad \mathit{fink}(t)\overset{\text{def}}{=}\mathit{fin}(t)\cap\mathcal{K};\quad \mathit{bink}(t)\overset{\text{def}}{=}\mathit{bin}(t)\cap\mathcal{K};\quad \mathit{fnk}(t)\overset{\text{def}}{=}\mathit{fn}(t)\cap\mathcal{K};\quad \mathit{bnk}(t)\overset{\text{def}}{=}\mathit{bn}(t)\cap\mathcal{K};\quad \mathit{nk}(t)\overset{\text{def}}{=}n(t)\cap\mathcal{K};$$
$$\mathit{onk}(t)\overset{\text{def}}{=}\mathit{on}(t)\cap\mathcal{K};\quad \mathit{fonk}(t)\overset{\text{def}}{=}\mathit{fon}(t)\cap\mathcal{K};\quad \mathit{bonk}(t)\overset{\text{def}}{=}\mathit{bon}(t)\cap\mathcal{K};$$

Definition 2−7: In a lock $L\overset{\text{def}}{=}\check{\kappa}@J$, κ is the *key* of L, denoted by function $\mathit{key}(L)\overset{\text{def}}{=}\kappa$, and J is the *locked polars set* of L, denoted by function $\mathit{lset}(L)\overset{\text{def}}{=}J$.

Abbreviations $\mathit{ikey}(L)\overset{\text{def}}{=}\mathit{inp}(\mathit{key}(L))$ and $\mathit{okey}(L)\overset{\text{def}}{=}\mathit{outp}(\mathit{key}(L))$. That is, $\mathit{ikey}(\check{\kappa}@J)\equiv\check{\kappa}$ and $\mathit{okey}(\check{\kappa}@J)\equiv\hat{\kappa}$.

Definition 2−8: For a choice branch of either form $B\overset{\text{def}}{=}!{}^{\dagger}m({}^{\dagger}\tilde{x})L.P$ or $B\overset{\text{def}}{=}!(\nu\,\kappa){}^{\dagger}m({}^{\dagger}\tilde{x})L.P$, we define

the *guard* of B: $\mathit{guard}(B)\overset{\text{def}}{=}m;$
the *locke* of B: $\mathit{locker}(B)\overset{\text{def}}{=}L;$

the *locke key* of B:	$\mathit{lkey}(B) \overset{\text{def}}{=} \mathit{key}(L)$;
the *exclusion se* of B:	$\mathit{excl}(B) \overset{\text{def}}{=} \mathit{lset}(L)$;
the *body* of B:	$\mathit{body}(B) \overset{\text{def}}{=} P$;
the *prefix* of B:	$\mathit{prefix}(B) \overset{\text{def}}{=} {}^{\dagger}\!m(\tilde{x})L$.

When $B=\mathbf{0}_\mathbf{G}$, we define $\mathit{body}(\mathbf{0}_\mathbf{G}) \overset{\text{def}}{=} \mathbf{0}_\mathbf{P}$, $\mathit{excl}(\mathbf{0}_\mathbf{G}) \overset{\text{def}}{=} \varnothing$, and $\mathit{prefix}(\mathbf{0}_\mathbf{G})$, $\mathit{guard}(\mathbf{0}_\mathbf{G})$, $\mathit{locker}(\mathbf{0}_\mathbf{G})$ and $\mathit{lkey}(\mathbf{0}_\mathbf{G})$ are undefined.

For a choice term, we define the functions

the *branches set*:	$\mathit{branch}(\mathbf{0}_\mathbf{G}) \overset{\text{def}}{=} \varnothing$, $\quad \mathit{branch}(B) \overset{\text{def}}{=} \{B\}$, $\quad \mathit{branch}(G_1 \otimes G_2) \overset{\text{def}}{=} \mathit{branch}(G_1) \cup \mathit{branch}(G_2)$;
the *guards set*:	$\mathit{guards}(G) \overset{\text{def}}{=} \{\mathit{guard}(B) : B \in \mathit{guard}(G)\}$;
the *lockers*:	$\mathit{lockers}(G) \overset{\text{def}}{=} \{\mathit{locker}(B) : B \in \mathit{guard}(G)\}$;
the *excludible set*:	$\mathit{excl}(B \otimes G) \overset{\text{def}}{=} \mathit{excl}(B) \cup \mathit{excl}(G)$;
the *arity*:	$\mathit{arity}(G) \overset{\text{def}}{=} \mathit{arity}(\mathit{branch}(G))$.

3 Semantics of locking scheme

In $\Lambda^{\circ}[G]$, the Λ term describes locking status in an independent language, and abstracts away other aspects of the process. The Λ term acts as a state machine (finite or infinite, depends on the grammar of the Λ language), and can be changed by adding and removing of a lock.

Adopt different syntaxes/semantics for the Λ language will introduce different locking scheme and locking status evolusion path, but will not interfare with semantic or syntax of the G language, or vice versa. However, in order to integrate the locking state calculus (Λ language) with the communication calculus, we need to predefine a syntax guideline and some function interfaces. The actual context of these functions depend on the syntax and semantics of the Λ language, and therefore the formal version of their definitions are delayed to the later part of this section.

The properties of these Λ language functions described in this section may not be interested theirselves, but are necessary later usage.

3.1 Generical properties and guideline

Definition 3−9: Function $\mathit{lock}(J,\kappa,\Lambda)$ gives the truth value to indicate whether "all polars in set J are locked by key κ in the locking list Λ".

While the middle parameter, the key κ, is omitted, we define $\mathit{lock}(J,\Lambda) \overset{\text{def}}{=} \forall {}^{\dagger}\!m \in J.\, \exists\kappa.\, (\mathit{lock}({}^{\dagger}\!m,\kappa,\Lambda))$ to indicate if "all polars in set J are locked by some key κ in the locking list Λ".

Definition 3−10: A Λ term is an empty locking list, denoted as $\Lambda=\lfloor\,\rfloor$, if $\mathit{lock}(J,\Lambda)$ is false for all $J\neq\varnothing$.

Addition to the above definitions, we define $\mathit{lock}(\varnothing,\kappa,\Lambda)\equiv$true (and therefore $\mathit{lock}(\varnothing,\Lambda)\equiv$true), even when $\Lambda=\lfloor\,\rfloor$. This is consistent with the convenience $\varnothing\subseteq\varnothing$ in the Set Theory, and can reduce the un-necessary steps of checking $J=\varnothing$.

Definition 3−11: Function $\mathit{lset}(\Lambda) \overset{\text{def}}{=} \{{}^{\dagger}\!m : \exists\kappa.(\mathit{lock}({}^{\dagger}\!m,\kappa,\Lambda)\}$ gives the set of all the polars which are locked by Λ.
Function $\mathit{keys}(\Lambda) \overset{\text{def}}{=} \{\kappa : \exists {}^{\dagger}\!m.(\mathit{lock}({}^{\dagger}\!m,\kappa,\Lambda)\}$, the set of all the keys which lock some polars in Λ.

Corollary 3−12: $\mathit{lset}(\lfloor\,\rfloor)\equiv\varnothing$ and $\mathit{keys}(\lfloor\,\rfloor)\equiv\varnothing$.

Lemma 3−13: $\mathit{lock}(J,\Lambda)\equiv(J\subseteq\mathit{lset}(\Lambda))$.
Proof: First, it is obversely true for the case $J=\varnothing$, since both $\mathit{lock}(\varnothing,\Lambda)$ and $\varnothing\subseteq\mathit{lset}(\Lambda)$ are true for all Λ.

Second, assume an abutrary J_1 satisfying $\mathit{lock}(J_1,\Lambda)=(J_1\subseteq \mathit{lset}(\Lambda))$, and let $J=J_1\cup\{{}^{\dagger}m\}$. By Definition-3-9, it must be $\mathit{lock}(J,\Lambda)\equiv \mathit{lock}(J_1,\Lambda)\wedge \mathit{lock}({}^{\dagger}m,\Lambda)\equiv(J_1\subseteq \mathit{lset}(\Lambda))\wedge \mathit{lock}({}^{\dagger}m,\Lambda)$.

Again by Definition-3-9, $\mathit{lock}({}^{\dagger}m,\Lambda)=\exists\kappa.\,\mathit{lock}({}^{\dagger}m,\kappa,\Lambda)\}$, then by Definition-3-11, $\mathit{lset}(\Lambda)=\{{}^{\dagger}m:\mathit{lock}({}^{\dagger}m,\Lambda)\}$, that is, $\mathit{lock}({}^{\dagger}m,\Lambda)\equiv({}^{\dagger}m\in \mathit{lset}(\Lambda))$. Therefore $\mathit{lock}(J,\Lambda)\equiv(J_1\subseteq \mathit{lset}(\Lambda))\wedge({}^{\dagger}m\in \mathit{lset}(\Lambda))\equiv(J\subseteq \mathit{lset}(\Lambda))$. ■

Definition 3−14: The $\mathit{addl}(L,\Lambda)$ and $\Lambda/\kappa@J$ are the basic operators for manipulation of Λ term:

$\mathit{addl}(L,\Lambda)$: adds lock L to Λ, such that it guarantees $\mathit{lock}(\mathit{lset}(L),\mathit{key}(L),\mathit{addl}(L,\Lambda))\equiv$true. This operation satisfies the constraint: If $\mathit{lset}(L)=\varnothing$ then $\mathit{addl}(L,\Lambda)=\Lambda$; If $\mathit{lset}(L)\neq\varnothing$ then for all J and κ,
$\mathit{lock}(J,\kappa,\mathit{addl}(L,\Lambda))\equiv \mathit{lock}(J,\kappa,\Lambda)\vee(\kappa=\mathit{key}(L)\wedge(J\subseteq \mathit{lset}(L)\vee \mathit{lock}(J-\mathit{lset}(L),\kappa,\Lambda)))$.

Λ/L : removes a lock of key $\kappa=\mathit{key}(L)$ (if any) from Λ for all polar ${}^{\dagger}m\in \mathit{lset}(L)$. It satisfies the following constraints:

$\lfloor\rfloor/L\equiv\lfloor\rfloor$;	$\mathit{addl}(L_1,\Lambda)/L\equiv \mathit{addl}(L_1,\Lambda/L)$, if $\mathit{key}(L_1)\neq \mathit{key}(L)$;
$\Lambda/\check{\kappa}@\varnothing\equiv\Lambda$;	$\mathit{addl}(\check{\kappa}@J,\Lambda/\check{\kappa}@J)\equiv\Lambda$, if $\mathit{lock}(J,\kappa,\Lambda)$;
$\Lambda/\check{\kappa}@(J\cup\{{}^{\dagger}m\})\equiv\Lambda/\check{\kappa}@J$, if $\neg \mathit{lock}({}^{\dagger}m,\kappa,\Lambda)$;	$\mathit{addl}(\check{\kappa}@J,\Lambda)/\check{\kappa}@J\equiv\Lambda$, if $\forall{}^{\dagger}m\in J.(\neg \mathit{lock}({}^{\dagger}m,\kappa,\Lambda))$.

Corollary 3−15: $\mathit{lset}(\mathit{addl}(L,\Lambda))\equiv \mathit{lset}(L)\cup \mathit{lset}(\Lambda)$; $\mathit{keys}(\mathit{addl}(L,\Lambda))\equiv\{\mathit{key}(L)\}\cup \mathit{keys}(\Lambda)$ if $\mathit{lset}(L)\neq\varnothing$.

Proof: By Lemma-3-13 and the definition of $\mathit{addl}(L,\Lambda)$ and $\mathit{lset}(\Lambda)$, we have

$$\begin{aligned}
\mathit{lset}(\mathit{addl}(L,\Lambda)) &\equiv\{{}^{\dagger}m:\exists\kappa.(\mathit{lock}({}^{\dagger}m,\kappa,\mathit{addl}(L,\Lambda))\}\\
&\equiv\{{}^{\dagger}m:\exists\kappa.(\mathit{lock}({}^{\dagger}m,\kappa,\Lambda)\vee(\kappa=\mathit{key}(L)\wedge({}^{\dagger}m\in \mathit{lset}(L)\vee \mathit{lock}(\{{}^{\dagger}m\}-\mathit{lset}(L),\kappa,\Lambda))))\}\\
&\equiv\{{}^{\dagger}m:{}^{\dagger}m\in \mathit{lset}(\Lambda)\vee\exists\kappa.(\kappa=\mathit{key}(L)\wedge({}^{\dagger}m\in \mathit{lset}(L)\vee \mathit{lock}(\{{}^{\dagger}m\}-\mathit{lset}(L),\kappa,\Lambda)))\}\\
&\equiv\{{}^{\dagger}m:{}^{\dagger}m\in \mathit{lset}(\Lambda)\vee{}^{\dagger}m\in \mathit{lset}(L)\vee \mathit{lock}(\{{}^{\dagger}m\}-\mathit{lset}(L),\kappa,\Lambda)\}\\
&\equiv\{{}^{\dagger}m:{}^{\dagger}m\in \mathit{lset}(L)\vee{}^{\dagger}m\in \mathit{lset}(\Lambda)\}\\
&\equiv \mathit{lset}(L)\cup \mathit{lset}(\Lambda)
\end{aligned}$$

When $\mathit{lset}(L)\neq\varnothing$, we have

$$\begin{aligned}
\mathit{keys}(\mathit{addl}(L,\Lambda)) &\equiv\{\kappa:\exists{}^{\dagger}m.(\mathit{lock}({}^{\dagger}m,\kappa,\mathit{addl}(L,\Lambda))\}\\
&\equiv\{\kappa:\exists{}^{\dagger}m.(\mathit{lock}({}^{\dagger}m,\kappa,\Lambda)\vee(\kappa=\mathit{key}(L)\wedge({}^{\dagger}m\in \mathit{lset}(L)\vee \mathit{lock}(\{{}^{\dagger}m\}-\mathit{lset}(L),\kappa,\Lambda))))\}\\
&\equiv\{\kappa:\kappa\in \mathit{keys}(\Lambda)\vee\exists{}^{\dagger}m.(\kappa=\mathit{key}(L)\wedge({}^{\dagger}m\in \mathit{lset}(L)\vee \mathit{lock}(\{{}^{\dagger}m\}-\mathit{lset}(L),\kappa,\Lambda)))\}\\
&\equiv\{\kappa:\kappa\in \mathit{keys}(\Lambda)\vee(\kappa=\mathit{key}(L))\}\\
&\equiv \mathit{keys}(\Lambda)\cup\{\mathit{key}(L)\}
\end{aligned}$$

■

Definition 3−16: The Λ terms Λ_1 and Λ_2 are said to be equivelant, denoted as $\Lambda_1\equiv\Lambda_2$, iff, $\mathit{lock}(J,\kappa,\Lambda_1)=\mathit{lock}(J,\kappa,\Lambda_2)$ for all κ and J, and $\mathit{addl}(L,\Lambda_1)\equiv \mathit{addl}(L,\Lambda_2)$ and $\Lambda_1/L\equiv\Lambda_2/L$ for all L.

Corollary 3−17: $\Lambda\equiv\Lambda'$ implies $\mathit{lset}(\Lambda)=\mathit{lset}(\Lambda')$, and $\mathit{keys}(\Lambda)=\mathit{keys}(\Lambda')$.

Now we can give the generic guideline for the of Λ term syntax:

Guideline 3−18: In any locking scheme, the syntax of Λ term should be able to be mapped to

$$\Lambda::=\lfloor\rfloor\ \big|\ \mathit{addl}(L,\Lambda)\ \big|\ \Lambda/L$$

and should saisfies the following constraints:

$\mathit{addl}(L_2,\mathit{addl}(L_1,\Lambda))\equiv \mathit{addl}(L_1,\mathit{addl}(L_2,\Lambda))$	and	$(\Lambda/L_1)/L_2\equiv(\Lambda/L_2)/L_1$	for all L_1 and L_2;
$\mathit{addl}(\check{\kappa}@(J_1\cup J_2),\Lambda)\equiv \mathit{addl}(\check{\kappa}@J_1,\mathit{addl}(\check{\kappa}@J_2,\Lambda))$	and	$\Lambda/\check{\kappa}@(J_1\cup J_2)\equiv(\Lambda/\check{\kappa}@J_1)/\check{\kappa}@J_2$	if $J_1\cap J_2=\varnothing$;

Definition 3−19: We also pre-define some abbreviation of operations:

Λ/κ removes from Λ any locker of key κ, and therefore satisfies $\forall\kappa'\neq\kappa.\forall J.(\mathit{lock}(J,\kappa',\Lambda/\kappa)\equiv \mathit{lock}(J,\kappa',\Lambda))$ and $\forall m.(\neg \mathit{lock}(\{{}^{\dagger}m\},\kappa,\Lambda/\kappa))$. It is an abbreviation of the procedure repeatly applies $/\kappa@\boldsymbol{M}$, and can be defined by:

$$\Lambda/\kappa\overset{\text{def}}{\equiv}\begin{cases}\lfloor\rfloor, & \text{if } \Lambda=\lfloor\rfloor;\\ \Lambda/\check{\kappa}@\boldsymbol{M}, & \text{if } \forall J.(\neg \mathit{lock}(J,\kappa,\Lambda/\kappa@\boldsymbol{M}));\\ (\Lambda/\check{\kappa}@\boldsymbol{M})/\kappa, & \text{otherwise.}\end{cases}$$

$\Lambda/@J$	removes from Λ all locks on any polar $\dagger m\in J$, and is an abbreviation of repeatly applies $/\breve{\kappa}@J$ operation for all $\kappa\in \mathit{keys}(\Lambda)$ until satisfying $\forall m.((\dagger m\in J\wedge\neg \mathit{lock}(\dagger m,\Lambda/@J))\vee(\dagger m\notin J\wedge \mathit{lock}(\dagger m,\Lambda/@J)\equiv \mathit{lock}(\dagger m,\Lambda))$. It can be defined by $\lfloor\rfloor/@J\stackrel{\text{def}}{=}\lfloor\rfloor$; $\Lambda/@\varnothing\stackrel{\text{def}}{=}\Lambda$; $\mathit{addl}(L,\Lambda)/@J\stackrel{\text{def}}{=}\mathit{addl}(\mathit{ikey}(L)@(\mathit{lset}(L)-J),\Lambda/@J)$.
$\Lambda\vert J$	removes from Λ all locks on any $\dagger m\notin J$. It is definied as the abbreviation $\Lambda\vert J\stackrel{\text{def}}{=}\Lambda/@(\mathit{lset}(\Lambda)-J)$.

We call $\Lambda|J$ the *significant part of Λ in domain J*, or $\mathit{sig}(\Lambda,J)$ for short, and $\Lambda/@J$ the *insignificant part of Λ in domain J*, or $\mathit{insig}(\Lambda,J)$ for short. If $\Lambda\equiv\mathit{sig}(\Lambda,J)$, then Λ is said to be *proper* (or *canonical*?) for *domain J*.

Corollary 3−20: The following properties of Λ term manipulations have been concluded.

1. $\Lambda/\breve{\kappa}@(J\cup J')\equiv\Lambda/\breve{\kappa}@J$, if $\forall\dagger m\in J'.(\neg \mathit{lock}(\dagger m,\kappa,\Lambda))$.
2. $\Lambda\equiv\Lambda/\breve{\kappa}@J$, if $J\cap \mathit{lset}(\Lambda)=\varnothing$ or $\kappa\notin \mathit{keys}(\Lambda)$.
3. $\Lambda\equiv \mathit{addl}(\breve{\kappa}@J',\Lambda/\breve{\kappa}@J)$, where $J'\stackrel{\text{def}}{=}\{\dagger m:\dagger m\in J\wedge \mathit{lock}(\{\dagger m\},\kappa,\Lambda)\}$.
4. $\Lambda/@(\mathit{lset}(\Lambda)\cup J)\equiv\lfloor\rfloor$ for all J.
5. $\Lambda|\varnothing\equiv\lfloor\rfloor$.
6. $\mathit{addl}(L,\Lambda)|J\equiv\Lambda|J$, if $\mathit{lset}(L)\cap J=\varnothing$.
7. $\mathit{lset}(\Lambda/@J)\equiv \mathit{lset}(\Lambda)-J$.
8. $\mathit{lset}(\Lambda|J)\equiv \mathit{lset}(\Lambda)\cap J$.
9. $(\Lambda/@J_1)/@J_2\equiv\Lambda/@(J_1\cup J_2)$.
10. $(\Lambda/L)/@J\equiv(\Lambda/@J)/L$.
11. $\mathit{addl}(L,\Lambda)|J\equiv \mathit{addl}(L,\Lambda|J)|J$.
12. $(\Lambda/L)|J\equiv(\Lambda|J)/L$.

Proof: 1. Let $J_1=\{\dagger m\}$ and $\mathit{lock}(\{\dagger m\},\kappa,\Lambda)$ is false, then by the definition of $\Lambda/\kappa@J$, we have $\Lambda/\kappa@(J\cup\{\dagger m\})\stackrel{\text{def}}{=}\Lambda/\kappa@J$. Assume arbitary J_k satisfying $\Lambda/\breve{\kappa}@(J\cup J_k)\equiv\Lambda/\breve{\kappa}@J$ and $\mathit{lock}(\{\dagger m\},\kappa,\Lambda)\equiv$false for all $\dagger m\in J_k$, then for $J'=J_k\cup\{\dagger m\}$ where $\mathit{lock}(\{\dagger m\},\kappa,\Lambda)\equiv$false, we have $\Lambda/\breve{\kappa}@(J\cup J')\equiv\Lambda/\kappa@(J\cup J_k\cup\{\dagger m\})\equiv\Lambda/\breve{\kappa}@(J\cup J_k)\equiv\Lambda/\breve{\kappa}@J$.

2. If $J\cap \mathit{lset}(\Lambda)=\varnothing$ or $\kappa\notin \mathit{keys}(\Lambda)$, then $\mathit{lock}(\{\dagger m\},\kappa,\Lambda)\equiv$false for all $\dagger m\in J$, by 1 and the definition of $\Lambda/\breve{\kappa}@J$ we have $\Lambda/\breve{\kappa}@J\equiv\Lambda/\breve{\kappa}@(\varnothing\cup J)\equiv\Lambda/\breve{\kappa}@\varnothing\equiv\Lambda$.

3. First, it is easy to verify the cases where $\Lambda=\lfloor\rfloor$ or $J=\varnothing$. For other cases, let $J''\stackrel{\text{def}}{=}\{\dagger m:\dagger m\in J\wedge\neg \mathit{lock}(\{\dagger m\},\kappa,\Lambda)\}$, then $J\equiv J'\cup J''$ and $J'\cap J''=\varnothing$. If $J''=\varnothing$, then $J\equiv J'$; if $J''\neq\varnothing$, then $\Lambda/\breve{\kappa}@J\equiv\Lambda/\breve{\kappa}@J'$ by 1 in this Corollary. In either cases, we have $\mathit{addl}(\breve{\kappa}@J',\Lambda/\breve{\kappa}@J)\equiv \mathit{addl}(\breve{\kappa}@J',\Lambda/\breve{\kappa}@J')\equiv\Lambda$, by the definition of Λ/L.

4. By the postcondition in the definition of $\Lambda/@J$, we have $\mathit{lock}(\dagger m,\Lambda/@(\mathit{lset}(\Lambda)\cup J)\equiv$false for all $\dagger m\in(\mathit{lset}(\Lambda)\cup J)$, by 7 which we will prove shortly, $\mathit{lset}(\Lambda/@(\mathit{lset}(\Lambda)\cup J)\equiv \mathit{lset}(\Lambda)-(\mathit{lset}(\Lambda)\cup J)\equiv\varnothing$. Then from Lemma-3-13, whenever $J\neq\varnothing$, we have $\mathit{lock}(J,\Lambda/@(\mathit{lset}(\Lambda)\cup J))\equiv$false, and by the definition of $\lfloor\rfloor$, $\Lambda/@(\mathit{lset}(\Lambda)\cup J)\equiv\lfloor\rfloor$.

5. $\Lambda|\varnothing\equiv\Lambda/@(\mathit{lset}(\Lambda)-\varnothing)\equiv\Lambda/@\mathit{lset}(\Lambda)$, then by 4., we have $\Lambda/@\mathit{lset}(\Lambda)\equiv\Lambda/@(\mathit{lset}(\Lambda)\cup\varnothing)\equiv\lfloor\rfloor$.

6. By $\mathit{lset}(L)\cap J=\varnothing$ and Lemma-3-13, we have $\mathit{lock}(J,\ \mathit{addl}(L,\Lambda))\equiv(J\subseteq \mathit{lset}(\Lambda))$. Therefore

$$\begin{aligned}\mathit{addl}(L,\Lambda)|J &\equiv \mathit{addl}(L,\Lambda)/@(\mathit{lset}(\mathit{addl}(L,\Lambda))-J)\\ &\equiv \mathit{addl}(L,\Lambda)/@(\mathit{lset}(\Lambda)\cup \mathit{lset}(L)-J)\\ &\equiv \mathit{addl}(L,\Lambda)/@((\mathit{lset}(\Lambda)-J)\cup \mathit{lset}(L))\end{aligned}$$

By the definition of the abbreviation $/@J$, this must include a step $/\mathit{key}(L)@\mathit{lset}(L)$, therefore by the definition of $\mathit{addl}(L,\Lambda)$ and 2., we have $\mathit{addl}(L,\Lambda)|J\equiv\Lambda/@(\mathit{lset}(\Lambda)\cup \mathit{lset}(L)-J)\equiv\Lambda/@(\mathit{lset}(\Lambda)-J)\equiv\Lambda|J$.

7. Apply Lemma-3-13 to $\forall m.((\dagger m\in J\wedge\neg \mathit{lock}(\{\dagger m\},\Lambda/@J))\vee(\dagger m\notin J\wedge(\mathit{lock}(\{\dagger m\},\Lambda/@J)\equiv \mathit{lock}(\{\dagger m\},\Lambda))))$, the postcondition in the definition of $\Lambda/@J$, then we have $J\cap \mathit{lset}(\Lambda/@J)\equiv\varnothing$ and $\mathit{lset}(\Lambda/@J)-J\equiv \mathit{lset}(\Lambda)-J$.

8. By 7. and the definition of $\Lambda|J$, $\mathit{lset}(\Lambda|J)\equiv \mathit{lset}(\Lambda)-(\mathit{lset}(\Lambda)-J)\equiv \mathit{lset}(\Lambda)\cap J$.

9. From the definition, it is easy to see $(\lfloor\rfloor/@J_1)/@J_2\equiv\lfloor\rfloor/@(J_1\cup J_2)$.
Assume it is true for some Λ', that is $(\Lambda'/@J_1)/@J_2\equiv\Lambda'/@(J_1\cup J_2)$, then by the definition of $\Lambda/@J$, we have

$$\begin{aligned}(\mathit{addl}(L,\Lambda')/@J_1)/@J_2&\equiv \mathit{addl}(\mathit{ikey}(L)@(\mathit{lset}(L)-J_1-J_2),\ (\Lambda'/@J_1)/@J_2)\\ &\equiv \mathit{addl}(\mathit{ikey}(L)@(\mathit{lset}(L)-(J_1\cup J_2)),\ \Lambda'/@(J_1\cup J_2))\\ &\equiv \mathit{addl}(L,\Lambda')/@(J_1\cup J_2).\end{aligned}$$

By inductiuon, $(\Lambda/@J_1)/@J_2\equiv\Lambda/@(J_1\cup J_2)$ for all Λ.

10. Since $\Lambda/@J$ is the abbrieviation of repeating $\Lambda/\breve{\kappa}_i@J$ for all $\kappa_i\in \mathit{keys}(\Lambda)$, therefore $(\Lambda/L)/@J\equiv(\Lambda/@J)/L$ can be conlcuded by repeating applying $(\Lambda/L_1)/L_2\equiv(\Lambda/L_2)/L_1$.

11. $$\begin{aligned}\mathit{addl}(L,\Lambda)|J &\equiv \mathit{addl}(L,\Lambda)/@(\mathit{lset}(L)\cup \mathit{lset}(\Lambda)-J)\\ &\equiv \mathit{addl}(\mathit{ikey}(L)@(\mathit{lset}(L)-(\mathit{lset}(L)\cup \mathit{lset}(\Lambda)-J)),\Lambda/@(\mathit{lset}(L)\cup \mathit{lset}(\Lambda)-J)) && \text{(definition)}\\ &\equiv \mathit{addl}(\mathit{ikey}(L)@(\mathit{lset}(L)\cap J)),\Lambda/@(\mathit{lset}(L)\cup \mathit{lset}(\Lambda)-J)).\end{aligned}$$

$$\begin{aligned} addl(L,\Lambda\,|\,J)\,|\,J &\equiv addl(L,\Lambda\,|\,J)/@(lset(L)\cup(lset(\Lambda)\cap J)-J)) \\ &\equiv addl(L,\Lambda\,|\,J)/@(lset(L)-J) \\ &\equiv addl(ikey(L)@(lset(L)-(lset(L)-J)),\ (\Lambda/@(lset(\Lambda)-J))/@(lset(L)-J)) && \text{(definition)} \\ &\equiv addl(ikey(L)@(lset(L)\cap J)),\Lambda/@(lset(L)\cup lset(\Lambda)-J)). \end{aligned}$$

12. Immediately concluded by applying 10 to the definition of $\Lambda\,|\,J$. ■

The clause 11 and 12 in the corollary is necessary for the rule **str-DISJ**, **tr_IN** and **tr_RELS** in the next session

Definition 3−21: The name functions for Λ terms are defined as $in(\Lambda)\overset{\text{def}}{=}lset(\Lambda)\cup keys(\Lambda)$, $bin(\Lambda)\overset{\text{def}}{=}\varnothing$, $on(\Lambda)\overset{\text{def}}{=}\varnothing$, $bon(\Lambda)\overset{\text{def}}{=}\varnothing$, $inc(\Lambda)\overset{\text{def}}{=}lset(\Lambda)$, $ink(\Lambda)\overset{\text{def}}{=}keys(\Lambda)$.

Definition 3−22: $\lfloor \check{\kappa}@{}^{\dagger}m \rfloor$ is called an *atom* of Λ, if $lock(\{{}^{\dagger}m\},\kappa,\Lambda)$. The function $atoms(\Lambda)\overset{\text{def}}{=}\{\lfloor \check{\kappa}@{}^{\dagger}m \rfloor : lock(\{{}^{\dagger}m\},\kappa,\Lambda)\}$ gives the set of all atoms of Λ.

Definition 3−23: When $lock(\{{}^{\dagger}m\},\Lambda)=\text{false}$, Λ does not block the polar ${}^{\dagger}m$, and we say that an action via ${}^{\dagger}m$ can *pass through* Λ, or Λ *allows the committment* on ${}^{\dagger}m$, denoted as $\Lambda\downarrow{}^{\dagger}m$. In contrast, when $lock(\{{}^{\dagger}m\},\Lambda)=\text{true}$ implies that channel m blocked by Λ, denoted as $\Lambda\not\downarrow{}^{\dagger}m$.

Given a set of channel names J and a key κ, if there exist some $J'\subseteq J$ such that $J'\neq\varnothing$ and $lock(J',\kappa,\Lambda)$, then it is said that the locking list Λ can commit on unlock signal $\check{\kappa}$ over J, denoted as $\Lambda\downarrow\check{\kappa}@J$, otherwise it is said that the Λ cannot commit on $\check{\kappa}$ over J, denoted as $\Lambda\not\downarrow\check{\kappa}@J$. When $J\supseteq lset(\Lambda)$, we may use $\Lambda\downarrow\check{\kappa}$ as the abbreviation of $\Lambda\downarrow\check{\kappa}@J$ to mean that there is some channel locked by κ in Λ, and use $\Lambda\not\downarrow\check{\kappa}$ as the abbreviation of $\Lambda\not\downarrow\check{\kappa}@J$ to mean that there is no channel locked by κ in Λ.

We denote ${}^{\dagger}m\uparrow L$ to mean that an input action via polar ${}^{\dagger}m$ proceeds, and triggers the lock L.

Clearly, $\lfloor\rfloor\downarrow{}^{\dagger}m$ for all ${}^{\dagger}m$.

Operation semantic for the Λ language is described by the following labelled transection rules:

$$\textbf{ltr\_LCK}:\ \frac{lock({}^{\dagger}m,\Lambda)=\text{false}}{\Lambda\xrightarrow{{}^{\dagger}m\uparrow L} addl(L,\Lambda)} \qquad\qquad \textbf{ltr\_ULK}:\ \frac{\kappa\in keys(\Lambda\,|\,J)}{\Lambda\xrightarrow{\check{\kappa}@J}\Lambda/\kappa@J}$$

Figure 3-1 Λ transection rules

Rule ltr_LCK is to say that, if the locking list Λ does not lock the polar ${}^{\dagger}m$, then an action via ${}^{\dagger}m$ can add to Λ the lock L triggered by that action.

Rule ltr_ULK means that, given a unlocking restriction set J, if there are some polarss in J locked by κ in the locking list Λ, then the receiving of the unlock signal $\check{\kappa}$ by Λ will cause the unlocking of all the channels in J locked by κ in Λ.

There is a problem we have to deal with when applying the Λ term into the communication calculus. Consider the GEC choice $\Lambda\circ[G]$, there is no restriction in either grammars can prevent in Λ some key κ lock some channel $a\notin guard(G)$. It may cause confusion if all the channels locked by κ in Λ are not members of $guard(G)$, for example, what could be the consequence if an unlocking signal $\hat{\kappa}$ is emitted to the air from some other corner of environment? To avoid this problem, we introduce some more notations.

3.2 A simple locking scheme (The set scheme)

In this scheme, one of the simplest, each channel can be locked by the same key only once, and any double locking by the same key will be treated as a single lock. In other words, a locking status Λ is recorded as a set of atoms. With the rules we adapted, a Λ term will act like the finite state machine. The Λ term grammar of this scheme is:

$$\Lambda ::= \lfloor\,\rfloor \mid \lfloor \tilde{L} \rfloor \mid \Lambda\,\Lambda$$

And the structural equivalencies, where locking status terms with structual equivalence are considered as the same, are shown in Figure 3-2:

lstr-SMM (Summation)	: $\lfloor\,\rfloor\Lambda \equiv \Lambda$;	$\Lambda_1\Lambda_2 \equiv \Lambda_2\Lambda_1$;		$\Lambda_1(\Lambda\Lambda_2) \equiv (\Lambda_1\Lambda)\Lambda_2$.
Lstr-EMP (Empty lock)	: $\lfloor \check{\kappa}@\varnothing \rfloor$	$\equiv \lfloor\,\rfloor$;		
Lstr-LKC (Combination)	: $\lfloor \check{\kappa}@J_1,\, \check{\kappa}@J_2 \rfloor$	$\equiv \lfloor \check{\kappa}@(J_1 \cup J_2) \rfloor$;		
Lstr-GRP (Grouping)	: $\lfloor \tilde{L}_1 \rfloor \lfloor \tilde{L}_2 \rfloor$	$\equiv \lfloor \tilde{L}_1, \tilde{L}_2 \rfloor$;		

Figure 3-2 Structual equivalence of locking status terms

The syntax of the communication calculus does not prevent the same key is used for locking multiple times on the same set of channels within the same *GEC* term, though it should be disencouraged for thread safe reason. The effect of multiple locking by the same key is handled in the locking state calculus by the lock combination rule Lstr-LKC. The chosen one in the above grammar is that "multiple locking by the same key should be unlocked by that key all at once". For example, $\lfloor \check{\kappa}@[{}^{\dagger}m_1,{}^{\dagger}m_2], \check{\kappa}@[{}^{\dagger}m_2,{}^{\dagger}m_3] \rfloor \equiv \lfloor \check{\kappa}@[{}^{\dagger}m_1,{}^{\dagger}m_2,{}^{\dagger}m_3] \rfloor$ and $\lfloor \check{\kappa}@J, \check{\kappa}@J \rfloor \equiv \lfloor \check{\kappa}@J \rfloor$. In other words, the underneath semantic of this scheme is that recording the lock status as a set of key-channel pairs. An alternative locking scheme, which we will describe in details in a separated paper, may record the lock status as a bag of key-channel pairs, and therefore in a Λ term the same channel name may be locked multiple times by the same key, and adding / remove a lock become increasing / decrease the counter for that particular lock, thus, Λ can have infinite states.

Definition 3−24: Function $lock(J, \kappa, \Lambda)$ introduced in Definition-3-9 is formally defined for this scheme by:
$lock(J, \kappa, \lfloor\,\rfloor) \overset{\text{def}}{=} (J{=}\varnothing)$; $\quad lock(J_1, \kappa_1, \lfloor \check{\kappa}_2@J_2 \rfloor \Lambda) \overset{\text{def}}{=} lock(J_1, \kappa_1, \Lambda)$; $\quad lock(J_1, \kappa, \lfloor \check{\kappa}@J_2 \rfloor \Lambda) \overset{\text{def}}{=} (J_1 \subseteq J_2) \vee lock(J_1{-}J_2, \kappa, \Lambda)$.

Corollary 3−25: By applying Definition-3-24 to Definition-3-11, then in this scheme, the function $lset(\Lambda)$ and $keys(\Lambda)$ can be equivalently calculated from:

$lset(\lfloor\,\rfloor) \equiv \varnothing$;	$lset(\lfloor \check{\kappa}@J \rfloor) \equiv J$;	$lset(\Lambda_1\Lambda_2) \equiv lset(\Lambda_1) \cup lset(\Lambda_2)$.
$keys(\lfloor\,\rfloor) \equiv \varnothing$;	$keys(\lfloor \check{\kappa}@J \rfloor) \equiv \{\kappa\}$, when $J \neq \varnothing$;	$keys(\Lambda_1\Lambda_2) \equiv keys(\Lambda_1) \cup keys(\Lambda_2)$.

Definition 3−26: The *element set* of Λ is $elems(\Lambda) \overset{\text{def}}{=} \{\check{\kappa}@J : J \neq \varnothing \wedge lock(J,\kappa,\Lambda) \wedge \forall {}^{\dagger}m \in (lset(\Lambda) - J).(\neg lock({}^{\dagger}m, \kappa, \Lambda)\}$.
A lock L is said to be an *element* of Λ if $L \in elems(\Lambda)$.

Clearly, $elems(\lfloor\,\rfloor) \overset{\text{def}}{=} \varnothing$, $\forall \Lambda \forall L, L' \in elems(\Lambda).(\ key(L) = key(L')$ implies $L = L'$).

Notation 3−27: Let $S_L = \{L_1, L_2, \ldots, L_n\}$ be a set of lock elements, we write $\Lambda \overset{\text{def}}{=} \lfloor S_L \rfloor$ to mean $\Lambda \overset{\text{def}}{=} \lfloor L_1, L_2, \ldots, L_n \rfloor$.

Definition 3−28: The *normal form* of a lock list Λ is defined as $norm(\Lambda) \overset{\text{def}}{=} \lfloor elems(\Lambda) \rfloor$.

Corollary 3−29: The following properties can be easily concluded:

1. $\Lambda \equiv norm(\Lambda)$ for all lock list Λ;
2. $elems(\lfloor\,\rfloor) \overset{\text{def}}{=} \varnothing$, $\forall \Lambda \forall L, L' \in elems(\Lambda).(\ key(L) = key(L')$ implies $L = L'$);
3. $atoms(\Lambda) \equiv \{ \lfloor \check{\kappa}@{}^{\dagger}m \rfloor : {}^{\dagger}m \in J \wedge (\check{\kappa}@J) \in elems(\Lambda) \}$;
4. $elems(\Lambda) \equiv \{ \check{\kappa}@J : \forall {}^{\dagger}m \in J.(\lfloor \check{\kappa}@{}^{\dagger}m \rfloor \in atoms(\Lambda)) \wedge \forall {}^{\dagger}m \notin J.(\lfloor \check{\kappa}@{}^{\dagger}m \rfloor \notin atoms(\Lambda))\ \}$;
5. $elems(\Lambda_1\Lambda_2) = \{ key(L)@(lset(L) \cup lset(L')) : L \in elems(\Lambda_1) \wedge L' \in elems(\Lambda_2) \wedge key(L) = key(L') \}$
$\cup \{ \quad L \quad : L \in (elems(\Lambda_1) \cup elems(\Lambda_2)) \wedge key(L) \notin (keys(\Lambda_1) \cap keys(\Lambda_2))\ \}$;

6. $\mathit{lset}(\Lambda)=\bigcup_{L\in \mathit{lset}(\Lambda)}\mathit{lset}(L)$, and $\mathit{keys}(\Lambda)=\{\mathit{key}(L): L\in \mathit{elems}(\Lambda)\}$;
7. $\Lambda\equiv\Lambda'$ iff $\mathit{elems}(\Lambda)=\mathit{elems}(\Lambda')$.

Definition 3−30: For this scheme, the operators introduced in Definition-3-14 can be formally defined as

$\mathit{addl}(L,\Lambda)$: $\mathit{addl}(L,\Lambda)\overset{\text{def}}{\equiv}\lfloor L\rfloor\,\Lambda$.

Λ/L : $\lfloor\check{\kappa}@J'\rfloor/\check{\kappa}@J\overset{\text{def}}{\equiv}\lfloor\check{\kappa}@(J'-J)\rfloor$; $\lfloor\check{\kappa}'@J'\rfloor/\check{\kappa}@J\overset{\text{def}}{\equiv}\lfloor\check{\kappa}'@J'\rfloor$, when $\kappa'\neq\kappa$; $(\Lambda_1\Lambda_2)/\check{\kappa}@J\overset{\text{def}}{\equiv}(\Lambda_1/\check{\kappa}@J)(\Lambda_2/\check{\kappa}@J)$.

Lemma 3−31: Definition-3-30 satisfies Definition-3-14.

Proof: $\mathit{addl}(L,\Lambda)$: $\mathit{lock}(\mathit{lset}(L),\mathit{key}(L),\mathit{addl}(L,\Lambda))\equiv\mathit{lock}(\mathit{lset}(L),\mathit{key}(L),\lfloor\mathit{key}(L)@\mathit{lset}(L)\rfloor\,\Lambda)$

$\equiv(\mathit{lset}(L)\subseteq\mathit{lset}(L))\vee\mathit{lock}(\varnothing,\mathit{key}(L),\Lambda)\equiv$true.

$\mathit{addl}(\check{\kappa}@\varnothing,\Lambda)\equiv\lfloor\kappa@\varnothing\rfloor\Lambda\equiv\lfloor\,\rfloor\Lambda\equiv\Lambda$;

$\mathit{lock}(J,\kappa,\mathit{addl}(L,\Lambda))\equiv(\kappa\neq\mathit{key}(L)\wedge\mathit{lock}(J,\kappa,\Lambda))\vee(\kappa=\mathit{key}(L)\wedge(J\subseteq\mathit{lset}(L)\vee\mathit{lock}(J-\mathit{lset}(L),\kappa,\Lambda)))$

$\equiv\mathit{lock}(J,\kappa,\Lambda)\vee(\kappa=\mathit{key}(L)\wedge(J\subseteq\mathit{lset}(L)\vee\mathit{lock}(J-\mathit{lset}(L),\kappa,\Lambda)))$;

Λ/L : $\lfloor\,\rfloor/\check{\kappa}@J\equiv\lfloor\check{\kappa}@\varnothing\rfloor/\check{\kappa}@J\equiv\lfloor\check{\kappa}@(\varnothing-J)\rfloor\equiv\lfloor\check{\kappa}@\varnothing\rfloor\equiv\lfloor\,\rfloor$;

$\Lambda/\check{\kappa}@\varnothing\equiv\lfloor\mathit{elems}(\Lambda)\rfloor/\check{\kappa}@\varnothing\equiv\Lambda$, since $\lfloor L_i\rfloor/\check{\kappa}@\varnothing\equiv\lfloor L_i\rfloor$; for each element $L_i\in\mathit{elems}(\Lambda)$;

$\Lambda/\check{\kappa}@(J\cup\{^{\dagger}m\})\equiv\Lambda/\check{\kappa}@J$ is true when $\kappa\notin\mathit{krys}(\Lambda)$, since $\forall J'.\forall L_i\in\mathit{elems}(\Lambda).(\lfloor L_i\rfloor/\check{\kappa}@J'\equiv\lfloor L_i\rfloor)$.

if $\kappa\in\mathit{krys}(\Lambda)$, then there must exist some J' such that $(\check{\kappa}@J')\in\mathit{elems}(\Lambda)$, therefore,

$\Lambda/\check{\kappa}@(J\cup\{^{\dagger}m\})\equiv(\lfloor\check{\kappa}@J'\rfloor\lfloor\mathit{elems}(\Lambda)-(\check{\kappa}@J')\rfloor)/\check{\kappa}@(J\cup\{^{\dagger}m\})\equiv(\lfloor\check{\kappa}@J'\rfloor/\check{\kappa}@(J\cup\{^{\dagger}m\}))\lfloor\mathit{elems}(\Lambda)-(\check{\kappa}@J')\rfloor)$,

but $^{\dagger}m\notin J'$ since $\neg\mathit{lock}(\{^{\dagger}m\},\kappa,\Lambda)$, therefore $\lfloor\check{\kappa}@J'\rfloor/\check{\kappa}@(J\cup\{^{\dagger}m\})\equiv\lfloor\check{\kappa}@(J'-(J\cup\{^{\dagger}m\}))\rfloor\equiv\lfloor\check{\kappa}@(J'-J)\rfloor$;

$\mathit{addl}(\check{\kappa}_1@J_1,\Lambda)/\kappa@J\equiv(\lfloor\check{\kappa}_1@J_1\rfloor/\kappa@J)(\Lambda/\kappa@J)\equiv\lfloor\check{\kappa}_1@J_1\rfloor(\Lambda/\check{\kappa}@J)\equiv\mathit{addl}(\check{\kappa}_1@J_1,\Lambda/\check{\kappa}@J)$;

If $\mathit{lock}(J,\kappa,\Lambda)$, then there must exist some J_1 such that $J\subseteq J_1$ and $\Lambda\equiv\lfloor\check{\kappa}@J_1\rfloor\lfloor\mathit{elems}(\Lambda)-(\check{\kappa}@J_1)\rfloor$.

That is, $\Lambda/\check{\kappa}@J\equiv(\lfloor\check{\kappa}@J_1\rfloor/\check{\kappa}@J)(\lfloor\mathit{elems}(\Lambda)-(\check{\kappa}@J_1)\rfloor/\check{\kappa}@J)\equiv\lfloor\check{\kappa}@(J_1-J)\rfloor\lfloor\mathit{elems}(\Lambda)-(\check{\kappa}@J_1)\rfloor$,

therefore $\mathit{addl}(\check{\kappa}@J,\Lambda/\check{\kappa}@J)\equiv\lfloor\check{\kappa}@J\rfloor\lfloor\check{\kappa}@(J_1-J)\rfloor\lfloor\mathit{elems}(\Lambda)-(\check{\kappa}@J_1)\rfloor\equiv\lfloor\mathit{elems}(\Lambda)\rfloor\equiv\Lambda$;

If $\forall^{\dagger}m\in J.(\neg\mathit{lock}(^{\dagger}m,\kappa,\Lambda))$, then $\Lambda/\check{\kappa}@J\equiv\Lambda$, therefore

$\mathit{addl}(\check{\kappa}@J,\Lambda)/\check{\kappa}@J\equiv(\lfloor\check{\kappa}@J\rfloor\Lambda)/\check{\kappa}@J\equiv(\lfloor\check{\kappa}@J\rfloor/\check{\kappa}@J)(\Lambda/\check{\kappa}@J)\equiv\Lambda$. ■

Corollary 3−32: In this scheme, some abbreviations introduced in Definition-3-19 can also be expressed as

$\Lambda/\kappa\equiv\lfloor\{L: L\in\mathit{elems}(\Lambda)\wedge\mathit{key}(L)\neq\kappa\}\rfloor$; $\Lambda/\kappa\equiv\Lambda/\kappa@\mathit{lset}(\Lambda)$; $\Lambda/\check{\kappa}@J\equiv\Lambda/\kappa$ when $J\supseteq\mathit{lset}(\Lambda)$;

$\Lambda/@J\equiv\lfloor\{\mathit{ikey}(L)@(\mathit{lset}(L)-J): L\in\mathit{elems}(\Lambda)\}\rfloor$; $\Lambda/@J\equiv(\ldots((\Lambda/\check{\kappa}_1@J)/\check{\kappa}_2@J)\ldots)/\check{\kappa}_n@J$ where $\mathit{keys}(\Lambda)=\{\kappa_1,\kappa_2,\ldots,\kappa_n\}$.

Corollary 3−33: The Guideline-3-18 is satified by this scheme. That is:

1. For all $\Lambda\neq\lfloor\,\rfloor$, there exist some Λ_1 and L where $\mathit{lset}(L)\neq\varnothing$, such that $\Lambda\equiv\mathit{addl}(L,\Lambda_1)$; and
2. For all Λ, there exist some Λ_1 and L such that $\Lambda\equiv\Lambda_1/L$; and
3. Satisfies constraints $\mathit{addl}(L_2,\mathit{addl}(L_1,\Lambda))\equiv\mathit{addl}(L_1,\mathit{addl}(L_2,\Lambda))$; $(\Lambda/L_1)/L_2\equiv(\Lambda/L_2)/L_1$;
   $\mathit{addl}(\check{\kappa}@(J_1\cup J_2),\Lambda)\equiv\mathit{addl}(\check{\kappa}@J_1,\mathit{addl}(\check{\kappa}@J_2,\Lambda))$ and $\Lambda/\check{\kappa}@(J_1\cup J_2)\equiv(\Lambda/\check{\kappa}@J_1)/\check{\kappa}@J_2$ if $J_1\cap J_2=\varnothing$.

Proof: The proof is trivial. ■

Lemma 3−34: For each Λ, J pair, there exist some $\Lambda'=\mathit{sig}(\Lambda,J)$ and $\Lambda''=\mathit{insig}(\Lambda,J)$ such that $\Lambda\equiv\Lambda'\Lambda''$.

Proof: If $\Lambda=\lfloor\,\rfloor$, then $\Lambda'\equiv\lfloor\,\rfloor$ and $\Lambda''\equiv\lfloor\,\rfloor$ for all J, therefore $\Lambda\equiv\Lambda'\Lambda''$.
If $\Lambda\neq\lfloor\,\rfloor$, let $\Lambda'=\lfloor\{\mathit{ikey}(L)@(\mathit{lset}(L)\cap J): L\in\mathit{elems}(\Lambda)\}\rfloor$ and $\Lambda''=\lfloor\{\mathit{ikey}(L)@(\mathit{lset}(L)-J): L\in\mathit{elems}(\Lambda)\}\rfloor$, then by Lstr-LKC, and by repeatly applying the commutation property in lstr-SMM, $\Lambda\equiv\Lambda'\Lambda''$. By the definition of *elems*, we have $(L\in\mathit{elems}(\Lambda))\equiv(\mathit{lset}(L)\neq\varnothing\wedge\mathit{lock}(\mathit{lset}(L),\mathit{key}(L),\Lambda)\wedge\forall^{\dagger}m\in(\mathit{lset}(\Lambda)-\mathit{lset}(L)).(\neg\mathit{lock}(\{^{\dagger}m\},\mathit{key}(L),\Lambda))$, therefore:

$\Lambda'\equiv\lfloor\{\mathit{ikey}(L)@(\mathit{lset}(L)-(\mathit{lset}(L)-J)): L\in\mathit{elems}(\Lambda)\}\rfloor$	(Set theory)
$\equiv\lfloor\{\mathit{ikey}(L)@\mathit{lset}(L): L\in\mathit{elems}(\Lambda)\}\rfloor/@(\mathit{lset}(\Lambda)-J)$	(Definition-3-30)
$\equiv\lfloor\mathit{elems}(\Lambda)\rfloor/@(\mathit{lset}(\Lambda)-J)$	(Definition of *elems*(Λ) and set theory)
$\equiv\Lambda/@(\mathit{lset}(\Lambda)-J)$	(Corollary-3-29)
$\equiv\mathit{sig}(\Lambda,J)$	(Definition)

And similarly we have

$\Lambda''\equiv\lfloor\{\mathit{ikey}(L)@\mathit{lset}(L): L\in\mathit{elems}(\Lambda)\}\rfloor/@J\equiv\Lambda/@J\equiv\mathit{insig}(\Lambda,J)$. ■

Corollary 3−35: For all Λ, J pairs, write $\Lambda'=\mathit{sig}(\Lambda,J)$ and $\Lambda''=\mathit{insig}(\Lambda,J)$, then $\Lambda/\kappa@J\equiv(\Lambda'/\kappa)\Lambda''$.

This corollary is necessary for the rule **str-DISJ** in the next session.

3.3 Another locking scheme (The bag scheme)

In an alternative locking scheme, if the same input polar of a channel is locked multiple times by the same key, then each of these duplicated locks have to be released separately before this channel available for input again. As an example, the effect of lock $\lfloor\check{\kappa}@[\bar{m}_1,\bar{m}_2],\check{\kappa}@[\bar{m}_2,\bar{m}_3]\rfloor$ is considered the same as that of $\lfloor\check{\kappa}@[\bar{m}_1,\bar{m}_2,\bar{m}_3],\check{\kappa}@\bar{m}_2\rfloor$, but different from $\lfloor\check{\kappa}@[\bar{m}_1,\bar{m}_2,\bar{m}_3]\rfloor$, since in the former the input polar of channel m_2 is locked twice, and has to be also unlocked twice to release that channel. Obviously, in this scheme Λ can have infinite states.

The Λ term grammar and the structural equivalencies rules for this scheme can be the same as the previous scheme, except a slightly different lock combination rule **Lstr-LKC**: $\lfloor\check{\kappa}@J_1\rfloor\lfloor\check{\kappa}@J_2\rfloor\equiv\lfloor\check{\kappa}@(J_1\cup J_2)\rfloor\lfloor\check{\kappa}@(J_1\cap J_2)\rfloor$.

We also adopt from the previous scheme the same definitions for function *lock*, *lset* and *keys*, but Λ term manipulation operators need to be redefined.

Definition 3−36: For this scheme, the operators introduced in Definition-3-14 can be formally defined as

$\mathit{addl}(L,\Lambda)$: $\mathit{addl}(L,\Lambda)\stackrel{\text{def}}{=}\lfloor L\rfloor\Lambda$.

Λ/L : $\lfloor\,\rfloor/\check{\kappa}@J\stackrel{\text{def}}{=}\lfloor\,\rfloor$; $\quad(\lfloor\check{\kappa}@J_1\rfloor\Lambda)/\check{\kappa}@J\stackrel{\text{def}}{=}\lfloor\check{\kappa}@(J_1-J)\rfloor(\Lambda/\check{\kappa}@(J-J_1))$; $\quad(\lfloor\check{\kappa}'@J'\rfloor\Lambda)/\check{\kappa}@J\stackrel{\text{def}}{=}\lfloor\check{\kappa}'@J'\rfloor(\Lambda/\check{\kappa}@J)$, if $\kappa'\neq\kappa$;

Corollary 3−37: In this scheme the following properties of Λ term manipulations have been concluded.

1. $\mathit{addl}(L_2,\mathit{addl}(L_1,\Lambda))\equiv\mathit{addl}(L_1,\mathit{addl}(L_2,\Lambda))$; 2. $\Lambda/\check{\kappa}@J_1/\check{\kappa}@J_2\equiv\Lambda/\check{\kappa}@(J_1\cup J_2)/\check{\kappa}@(J_1\cap J_2)$;
3. $(\Lambda/L_1)/L_2\equiv(\Lambda/L_2)/L_1$; 4. $\Lambda\equiv\mathit{addl}(\check{\kappa}@J,\Lambda)/\check{\kappa}@J$.

Proof: 1. The proof is trivial for 1.

2. Obviously, $\lfloor\,\rfloor/\check{\kappa}@J_1/\check{\kappa}@J_2\equiv\lfloor\,\rfloor/\check{\kappa}@(J_1\cup J_2)/\check{\kappa}@(J_1\cap J_2)$ for all κ, J_1 and J_2.

Let L be an arbitrary lock, and assume some Λ_n satisfying $(\Lambda_n/\check{\kappa}@J_1)/\check{\kappa}@J_2\equiv(\Lambda_n/\check{\kappa}@(J_1\cup J_2))/\check{\kappa}@(J_1\cap J_2)$ for all κ, J_1 and J_2. Lets write $J\stackrel{\text{def}}{=}\mathit{lset}(L)$, $J'\stackrel{\text{def}}{=}J_1\cup J_2$ and $J''\stackrel{\text{def}}{=}J_1\cap J_2$.

If $\mathit{key}(L)\neq\kappa$, then $(\lfloor L\rfloor\Lambda_n)/\check{\kappa}@J_1/\check{\kappa}@J_2\equiv\lfloor L\rfloor(\Lambda_n/\check{\kappa}@J_1/\check{\kappa}@J_2)\equiv\lfloor L\rfloor(\Lambda_n/\check{\kappa}@(J_1\cup J_2)/\check{\kappa}@(J_1\cap J_2))\equiv(\lfloor L\rfloor\Lambda_n)/\check{\kappa}@J'/\check{\kappa}@J''$.

If $\mathit{key}(L)=\kappa$, $(\lfloor L\rfloor\Lambda_n)/\check{\kappa}@J_1/\check{\kappa}@J_2\equiv\lfloor\check{\kappa}@((J-J_1)-J_2)\rfloor\Lambda_n/\check{\kappa}@(J_1-J)/\check{\kappa}@(J_2-(J-J_1))$. However, by set theory, we have $(J-J_1)-J_2\equiv J-(J_1\cup J_2)\equiv(J-(J_1\cup J_2))-(J_1\cap J_2)\equiv(J-J')-J''$,

$(J_1-J)\cup(J_2-(J-J_1))\equiv((J_1\cup J_2)-J)\cup(J_1\cap J_2)\equiv(J'-J)\cup J''$,

$(J_1-J)\cap(J_2-(J-J_1))\equiv((J_1\cup J_2)-J)\cap(J_1\cap J_2)\equiv(J'-J)\cap J''$,

$(J_1\cap J_2)\equiv(J_1\cap J_2)-(J-(J_1\cup J_2))\equiv J''-(J-J')$.

Therefore,
$$\begin{aligned}(\lfloor L\rfloor\Lambda_n)/\check{\kappa}@J_1/\check{\kappa}@J_2&\equiv\lfloor\check{\kappa}@((J-J')-J'')\rfloor\Lambda_n/\check{\kappa}@((J'-J)\cup J'')/\check{\kappa}@((J'-J)\cap J'')\\&\equiv\lfloor\check{\kappa}@((J-J')-J'')\rfloor\Lambda_n/\check{\kappa}@(J'-J)/\check{\kappa}@J''\\&\equiv\lfloor\check{\kappa}@((J-J')-J'')\rfloor\Lambda_n/\check{\kappa}@(J'-J)/\check{\kappa}@(J''-(J-J'))\\&\equiv(\lfloor L\rfloor\Lambda_n)/\check{\kappa}@J'/\check{\kappa}@J''.\end{aligned}$$

By induction, $\Lambda/\check{\kappa}@J_1/\check{\kappa}@J_2\equiv\Lambda/\check{\kappa}@(J_1\cup J_2)/\check{\kappa}@(J_1\cap J_2)$ for all Λ, κ, J_1 and J_2.

3. First, $(\lfloor\,\rfloor/L_1)/L_2\equiv(\lfloor\,\rfloor/L_2)/L_1$ for all L_1 and L_2.

Assume an arbitrary lock L and assume some Λ_n satisfying $(\Lambda_n/L_1)/L_2\equiv(\Lambda_n/L_2)/L_2$ for all L_1 and L_2, write $\kappa=\mathit{key}(L)$, $J=\mathit{lser}(L)$, $\kappa_1=\mathit{key}(L_1)$, $J_1=\mathit{lser}(L_1)$, $\kappa_2=\mathit{key}(L_2)$, $J_2=\mathit{lser}(L_2)$, then:

If $\kappa_1\neq\kappa$ and $\kappa_1\neq\kappa$, $((\lfloor L\rfloor\Lambda_n)/L_1)/L_2\equiv\lfloor L\rfloor((\Lambda_n/L_1)/L_2)\equiv\lfloor L\rfloor((\Lambda_n/L_2)/L_1)\equiv((\lfloor L\rfloor\Lambda_n)/L_2)/L_1$;

If $\kappa=\kappa_1=\kappa_2$, $((\lfloor L\rfloor\Lambda_n)/L_1)/L_2\equiv\lfloor\check{\kappa}@(J-J_1)-J_2)\rfloor((\Lambda_n/\check{\kappa}@(J_1-J)/\check{\kappa}@(J_2-(J-J_1)))$. However, by set theory, we have $(J-J_1)-J_2\equiv(J-J_2)-J_1$,

$J_2-(J-J_1)\equiv(J_2-J)\cup(J\cap J_1\cap J_2)$,

$(J_1-J)\cup(J_2-(J-J_1))\equiv(J_1-J)\cup(J_2-J)\cup(J\cap J_1\cap J_2)\equiv(J_1-J)\cup(J_2-(J-J_1))$,

$(J_1-J)\cap(J_2-(J-J_1))\equiv(J_2-J)\cap(J_1-(J-J_2))$;

$$\begin{aligned}
\text{Therefore, } (\lfloor L\rfloor \Lambda_n)/\check{\kappa}@J_1/\check{\kappa}@J_2 &\equiv \lfloor \check{\kappa}@((J-J_1)-J_2)\rfloor \Lambda_n/\check{\kappa}@((J_1-J)\cup(J_2-(J-J_1)))/\check{\kappa}@((J_1-J)\cap(J_2-(J-J_1)))\\
&\equiv \lfloor \check{\kappa}@((J-J_2)-J_1)\rfloor \Lambda_n/\check{\kappa}@(J_1-J)/\check{\kappa}@(J_1-(J-J_2))\\
&\equiv \lfloor \check{\kappa}@((J-J')-J'')\rfloor \Lambda_n/\check{\kappa}@(J'-J)/\check{\kappa}@(J''-(J-J'))\\
&\equiv (\lfloor L\rfloor \Lambda_n)/\check{\kappa}@J_2/\check{\kappa}@J_1;
\end{aligned}$$

$$\begin{aligned}
\text{If } \kappa=\kappa_1\neq\kappa_2,\quad ((\lfloor L\rfloor \Lambda_n)/L_1)/L_2 &\equiv \lfloor \check{\kappa}@(J-J_1)\rfloor ((\Lambda_n/\check{\kappa}@(J_1-J))/\check{\kappa}_2@J_2)\\
&\equiv \lfloor \check{\kappa}@(J-J_1)\rfloor ((\Lambda_n/\check{\kappa}_2@J_2)/\check{\kappa}@(J_1-J))\\
&\equiv ((\lfloor L\rfloor \Lambda_n)/L_2)/L_1;
\end{aligned}$$

Similar for $\kappa=\kappa_2\neq\kappa_1$.

By induction, $(\Lambda/L_1)/L_2\equiv(\Lambda/L_2)/L_1$ for all Λ, L_1 and L_2

4. $add(\check{\kappa}@J,\Lambda)/\check{\kappa}@J\equiv(\lfloor\check{\kappa}@J\rfloor\Lambda)/\check{\kappa}@J\equiv\lfloor\check{\kappa}@(J-J)\rfloor(\Lambda/\check{\kappa}@(J-J))\equiv\lfloor\check{\kappa}@\emptyset\rfloor(\Lambda/\check{\kappa}@\emptyset)\equiv\Lambda$. ■

Corollary 3−38: The Guideline-3-18 is satified by this scheme.
Proof: The proofs are either trivial or have been proved in the previous corollary. ■

The semantic of this scheme it is equivalent to record the lock status as a bag of key-channel pairs, and therefore the adding / removing a lock are equivalent to increasing / decreasing the counter for that particular lock in a Λ term.

4 Semantics of Processes Communication

Structural equivalencies: agents with structural equivalence are considered as the same

Summation		
str-SUM1:	$P_1 \mid \mathbf{0_P} \equiv P_1$;	$G_1\otimes \mathbf{0_G} \equiv G_1$;
str-SUM2:	$P_1 \mid P_2 \equiv P_2 \mid P_1$;	$G_1\otimes G_2 \equiv G_2\otimes G_1$;
str-SUM3:	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$;	$G_1\otimes(G_2\otimes G_3) \equiv (G_1\otimes G_2)\otimes G_3$
Null		
str-NUL:	$\Lambda\circ[\mathbf{0_G}] \equiv \mathbf{0_P}$	
str-DISJ:	$\Lambda\circ[G] \equiv (sig(\Lambda, guard(G)))\circ[G]$	
	$\Lambda\circ[\,!(\nu\,\kappa){}^{\dagger}m(\tilde{x})\check{\kappa}@J.P\otimes G] \equiv \Lambda\circ[!(\nu\,\kappa){}^{\dagger}m(\tilde{x})\check{\kappa}@(J\cap({}^{\dagger}m\cup guard(G)).P\otimes G]$	
Instance		
str-INS:	$((\tilde{w})P)\langle\tilde{a}\rangle \equiv P\{\tilde{a}/\tilde{w}\}$	$((\tilde{w})G)\langle\tilde{a}\rangle \equiv G\{\tilde{a}/\tilde{w}\}$
Scope		
str-SCP1:	$(\nu\,n)P\equiv P$, if $n\notin fn(P)$;	$(\nu\,n)G\equiv G$, if $n\notin fn(G)$;
		$!(\nu\,\kappa)\beta.P\equiv\,!\beta.P$, If $\kappa\notin fn(\beta.P)$
str-SCP2:	$(\nu\,n_1)(\nu\,n_2)P\equiv(\nu\,n_2)(\nu\,n_1)P$;	$(\nu\,n_1)(\nu\,n_2)P\equiv(\nu\,n_1,n_2)P$
str-SCP3:	$(\nu\,m)\overline{m}\langle\tilde{y}\rangle\equiv\mathbf{0_P}$;	$(\nu\,m)!{}^{\dagger}m(\tilde{x})L.P\equiv\mathbf{0_G}$;
	$(\nu\,\kappa)\hat{\kappa}\equiv\mathbf{0_P}$;	$(\nu\,\kappa)\Lambda\circ[G]\equiv(\nu\,\kappa)\Lambda\circ[\mathbf{0_G}]$, if $lock(guard(G),\kappa,\Lambda)$ is true;
		$(\nu\,\kappa)\Lambda\circ[G\otimes G']\equiv\Lambda\circ[G']$, if $lock(guard(G),\kappa,\Lambda)$ and $\kappa\notin fn(G')$
str-SCP4:	$(\nu\,n)P_1 \mid P_2\equiv(\nu\,n)(P_1 \mid P_2)$, if $n\notin fn(P_2)$;	$(\nu\,n)G_1\otimes G_2\equiv(\nu\,n)(G_1\otimes G_2)$, if $n\notin fn(G_2)$;
	$\Lambda\circ[(\nu\,\kappa)G]\equiv(\nu\,\kappa)\Lambda\circ[G]$, if $\kappa\notin keys(\Lambda)$	
str-REN:	$(\nu\,n_1)P\equiv(\nu\,n_2)(P\{n_2/n_1\})$, if $n_2\notin fn(P)$	

Figure 4-1 Structural congruence rules for the κ-calculus

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

Corollary 4−39: If $lset(L) \cap guard(G) = \varnothing$ then $addl(L,\Lambda)\circ[G] \equiv \Lambda\circ[G]$.
Proof: By the 6th clause in Corollary-3-20, $sig(addl(L,\Lambda), guard(G)) \equiv sig(\Lambda, guard(G))$, then apply to rule str-DISJ. ■

Operation semantic is described with labelled transection Reduction rules:

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

tr_SIG: $\dfrac{\bullet}{\hat{\kappa} \xrightarrow{\hat{\kappa}} \mathbf{0_P}}$

tr_IN: $\dfrac{\Lambda \xrightarrow{{}^{\dagger}m\uparrow L} \Lambda', \qquad P' = P\{^{\tilde{\bar{u}}}/_{\tilde{x}}\}}{\Lambda\circ[G \otimes !(\nu\,\kappa)\,{}^{\dagger}m(\tilde{x})L.P] \xrightarrow{{}^{\dagger}m(\tilde{\bar{u}})} (\nu\,\kappa)(P' \mid \Lambda'\circ[G \otimes !(\nu\,\kappa)\,{}^{\dagger}m(\tilde{x})L.P])}$

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

tr_PARL: $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$

tr_SYNC1: $\dfrac{P \xrightarrow{(\nu\tilde{v})\bar{m}\langle\tilde{\bar{u}}\rangle} P', \qquad P' \xrightarrow{{}^{\dagger}m(\tilde{\bar{u}})} P''}{(\nu\, m)P \xrightarrow{\tau} (\nu\, m)(\nu\,\tilde{v})P''}$

tr_SYNC2: $\dfrac{P \xrightarrow{\hat{\kappa}} P', \qquad P' \xrightarrow{\check{\kappa}} P''}{(\nu\,\kappa)P \xrightarrow{\tau} (\nu\,\kappa)P''}$

tr_RES: $\dfrac{P \xrightarrow{\alpha} P', \qquad \tilde{n} \cap fn(\alpha) = \varnothing}{(\nu\,\tilde{n})P \xrightarrow{\alpha} (\nu\,\tilde{n})P'}$

tr_STRUC: $\dfrac{P_1' \equiv P_1, \quad P_1 \xrightarrow{\alpha} P_2, \quad P_2 \equiv P_2'}{P_1' \xrightarrow{\alpha} P_2'}$

Figure 4-2 Labelled transition rules for process terms in the κ-calculus

Remark 4−40: Similar to the polar π-calculus, in the the κ-calculus the τ action is truly internal, that is, neither visible nor interruptible by external observers. Therefore, the name restrictions in rule tr_SYNC1 and tr_SYNC2 are required. Without it, the synchronisation will not be considered as an internal action, but a two steps action, such as $P \mid Q \xrightarrow{(\nu\tilde{v})\bar{m}\langle\tilde{\bar{u}}\rangle}.\xrightarrow{{}^{\dagger}m(\tilde{\bar{u}})} (\nu\,\tilde{v})\,(P' \mid Q')$ or $P \mid Q \xrightarrow{\hat{\kappa}}.\xrightarrow{\check{\kappa}} P' \mid Q'$, where both steps are visible for external observers. This strong requirement on τ actions is necessary for guaranteeing the standard rule $fn(\tau) = bn(\tau) = \varnothing$ ([Amadio96]) valid, and is necessary for preserving τ actions in output polars substitution.

Definition 4−41: (**Weak transitiont**): As usual, let ( )* represent that the contents in ( ) repeating zero or finitely many times, then the weak transitions are defined as: $P \overset{\tau}{\Longrightarrow} P'$ iff $P(\xrightarrow{\tau})^{*}P'$;
$P \overset{\alpha}{\Longrightarrow} P'$ iff $P \overset{\tau}{\Longrightarrow}.\xrightarrow{\alpha}.\overset{\tau}{\Longrightarrow} P'$, where $\alpha \neq \tau$.

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

Definition 4−42: $P \longrightarrow P'$ iff $(\nu\, m)P \xrightarrow{\tau} (\nu\, m)P'$ for some m; $\quad P \Longrightarrow P'$ iff $(\nu\, m)P \overset{\tau}{\Longrightarrow} (\nu\, m)P'$ for some m.

With this definition, we then can have a variant of the rule tr_SYNC1: $\dfrac{P \xrightarrow{(\nu\tilde{v})\bar{m}\langle\tilde{\bar{u}}\rangle} P', \quad Q \xrightarrow{{}^{\dagger}m(\tilde{\bar{u}})} Q', \quad \tilde{v} \cap fn(Q) = \varnothing}{P \mid Q \longrightarrow (\nu\,\tilde{v})\,(P' \mid Q')}$

Beside the reason we have just discussed, the distinguish between internal action τ and reduction is also necessary for the new bisimulation relation, responsive bisimulation, which we will discuss later in the next session.

Definition 4−43: The strong commitments are defined as:

Process P can *commit* the action α, denoted as $P\downarrow\alpha$, if there exists some P' such that $P \xrightarrow{\alpha} P'$.

Process P can *commit* on input polar ${}^{+}m$, denoted as $P\downarrow {}^{+}m$, if there exists some input action $\alpha = {}^{+}m(\tilde{u})$ s.t. $P\downarrow\alpha$;
Process P can *commit* on output polar $\bar{m}$, denoted as $P\downarrow\bar{m}$, if there is some output action $\alpha=(\nu\,\tilde{v})\bar{m}\langle\tilde{u}\rangle$ s.t. $P\downarrow\alpha$;

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

Definition 4−44: Process P' is a *descendant* of process P, iff there exists a finite *action sequence* $\ell=\alpha_1,\alpha_2,\ldots,\alpha_n$ such that $P\xrightarrow{\alpha_1}P_1, P_1\xrightarrow{\alpha_2}P_2, P_2\xrightarrow{\alpha_3}P_3, \ldots, P_{n-1}\xrightarrow{\alpha_n}P'$, or $P\xrightarrow{\ell}P'$ for short, and it is said that P commits on ℓ, denoted as $P\downarrow\ell$.

5 Processes Bisimulations

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

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

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

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

The barbed bisimulation ([Milner92b],[Sangiorgi92b]) is a rather weak relation, which traces the state changes of a process during the course of reductions, and observes which channels available for communication. As a polarised process calculus, in the κ-calculus only output polars (of both communication channels and locking keys) are considered as observable, therefore we adopt a version of barbed bisimulation similar to that in [Zhang02A] for the polar π-calculus.

Definition 5−45 (**barbed bisimulation**): A symmetric relation $\mathcal{S}$ on P-terms is a (strong) barbed bisimulation if whenever $P\mathcal{S}Q$ then $P\downarrow a$ implies $Q\downarrow a$ for all $a\in\bar{\mathcal{M}}$, and $P\longrightarrow P'$ implies $\exists Q'$ such that $Q\longrightarrow Q'$ and $P'\mathcal{S}Q'$.

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

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

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

Definition 5−46: In the κ-calculus, **process context** $\mathcal{C}[.]$ is given by $\mathcal{C}::=[.]\,\big|\,(\nu\,\tilde{n})\mathcal{C}\,\big|\,\mathcal{C}|P\,\big|\,\Lambda\circ\{!(\nu\kappa)\beta.\mathcal{C}\otimes G\}$.

Definition 5−47: Let $\mathcal{C}[.]$ be process context, then we define the barbed equivalences and their κ-versions as
strong and weak **barbed equivalence**: $P\simeq_{\mathrm{b}}Q$ if $\forall\mathcal{C}[.].(\mathcal{C}[P]\sim_{\mathrm{b}}\mathcal{C}[Q])$; $P\approxeq_{\mathrm{b}}Q$ if $\forall\mathcal{C}[Q].(\mathcal{C}[P]\approx_{\mathrm{b}}\mathcal{C}[Q])$;
strong and weak **barbed** ***κ*****-equivalence**: $P\simeq_{\kappa\mathrm{b}}Q$ if $\forall\mathcal{C}[.].(\mathcal{C}[P]\sim_{\kappa\mathrm{b}}\mathcal{C}[Q])$; $P\approxeq_{\kappa\mathrm{b}}Q$ if $\forall\mathcal{C}[Q].(\mathcal{C}[P]\approx_{\kappa\mathrm{b}}\mathcal{C}[Q])$.

Or, for the still weaker versions similar in [Amadio96], let R be arbitrary process, then we define that

strong and weak **barbed** 1-**equivalence**: $P\simeq_{\mathrm{b}1}Q$ if $\forall R.(R|P\sim_{\mathrm{b}}R|Q)$; $P\cong_{\mathrm{b}1}Q$ if $\forall R.(R|P\approx_{\mathrm{b}}R|P)$;
strong and weak **barbed** κ**1-equivalence**: $P\simeq_{\kappa\mathrm{b}1}Q$ if $\forall R.(R|P\sim_{\kappa\mathrm{b}}R|Q)$; $P\cong_{\kappa\mathrm{b}1}Q$ if $\forall R.(R|P\approx_{\kappa\mathrm{b}}R|P)$;

The object systems have the following characteristics:

1. Only the output actions performed by an object are observable from outside of that object;
2. The effects of an input action performed by an object can be observed only via consequent reactions (output) from that object;
3. After a message to an object is sent, it is not possible to know when or whether it will be received until a response message is returned from that object.

In other words, the behaviours of an object can be only detected by responses. However, as pointed out by [Zhang02A] and [Zhang02B], even the weak barbed equivalence is too strong for compositional objects. For example, let process $O_1\overset{\mathrm{def}}{=}(\nu\, n)(!\overset{\dagger}{m}(\tilde{x}).\bar{n}\langle\tilde{x}\rangle\,|\,\lfloor\check{\kappa}@[\overset{\dagger}{n}]\rfloor\circ\lceil !\overset{\dagger}{n}(\tilde{x})L.Body\rceil\ )$ and $O_2\overset{\mathrm{def}}{=}\lfloor\check{\kappa}@[\overset{\dagger}{m}]\rfloor\circ\lceil !\overset{\dagger}{m}(\tilde{x})L.Body\rceil$ express 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 the barbed 1-equivalence. The weak barbed equivalences fail in at least two ways:

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

Figure 5-1

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

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

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

Second, it cannot prevent input names clash between the testing environment and the processes being tested. For example, let $R\overset{\mathrm{def}}{=}\overset{\dagger}{m}(\tilde{x}).\bar{x}\langle\tilde{q}\rangle\,|\,\bar{m}\langle\tilde{p}\rangle$, then as shown in Figure 5-1, $O_1\,|\,R$ can take two different reduction paths:

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

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

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

We need a different technique to measure the observation behaviours, weak enough to ignore the unrelated information and strong enough to distinguish the similarity in responses perceived by outsiders. As with barbed bisimulation, we must note that the state changes of a process caused by internal actions, and we must also be able to detect which communication channels are available for output in all evolved states. What is more, in order to distinguish states, we need to be able to observe what each of the messages output by the process is. The οτ-bisimulation, defined in the same way as that in [Zhang02A], can provide this degree of observation:

Definition 5−48: The (strong) $o\tau$-**bisimulation** is a symmetric relation $\mathcal{S}$ on processes such that whenever $P\mathcal{S}Q$ then $P\xrightarrow{\alpha}P'$ implies $Q\xrightarrow{\alpha}Q'$ and $P'\mathcal{S}\,Q'$ for all action α in the form of either $\alpha=(\nu\,\tilde{v})\bar{m}\langle\tilde{u}\rangle$ or $\alpha=\tau$, and $bn(\alpha)\cap fn(Q)=\varnothing$.

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

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

Lemma 5−49: Each of κ-version and non-κ-version οτ-bisimulations, $\mathcal{S}$, is preserved by restriction, that is, $P\mathcal{S}Q$ implies $(\nu\,\tilde{n})P\mathcal{S}(\nu\,\tilde{n})Q$.

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

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

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

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

5.1 Responsive Bisimulation

Notation 5−50: We add the auxiliary P-term $[{}^{\dagger}m\langle\tilde{u}\rangle]P$, the *localisation* of the sent message $\bar{m}\langle\tilde{u}\rangle$ with process P, into the process syntax. Properties for this term are shown in Figure 5-2.

Structural equivalence :

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

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

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

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

Transition :

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

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

Figure 5-2 Localised output action.

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

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

The rule lTr_SYNC3 added a new case for defining the τ action. Unlike in rule tr_SYNC1, here is no name restriction is required. However, since only the input polar, ${}^{\dagger}m$, of the channel name m is involved, and the reservation of τ actions is maintained by input prefixing.

Corollary 5−51: The following conclusion can be immediately drew from the rules in Figure 5-2:

(1) If $P \xrightarrow{\overline{m}\langle\tilde{u}\rangle} P'$ then $(\nu\, m)P \equiv (\nu\, m)\,[{}^{\dagger}m\langle\tilde{u}\rangle]P'$;
(3) $P{\downarrow}{}^{\dagger}m$ implies $(\,[{}^{\dagger}m\langle\tilde{u}\rangle]P\,){\downarrow}\tau$;
(2) $P\not\downarrow\alpha$ implies $(\,[{}^{\dagger}m\langle\tilde{u}\rangle]P\,)\not\downarrow\alpha$ if $\alpha\neq\tau$, or, $\alpha=\tau$ but $P\not\downarrow{}^{\dagger}m$;
(4) $(\,[{}^{\dagger}m\langle\tilde{u}\rangle]P\,)\not\downarrow{}^{\dagger}m(\tilde{u})$.

Now we can begin to introduce new behaviour equivalence relations.

Definition 5−52: Let $\mathcal{T}[.]$ be the **responsive testing context** of syntax $\mathcal{T}::=[.]\,\big|\,[{}^{\dagger}m\langle\tilde{u}\rangle]\mathcal{T}$, then we define the strong and weak **responsive equivalence**: $P\simeq_r Q$ iff $\forall\mathcal{T}.(\mathcal{T}[P]\sim_{o\tau}\mathcal{T}[Q])$, $P\cong_r Q$ iff $\forall\mathcal{T}.(\mathcal{T}[P]\approx_{o\tau}\mathcal{T}[Q])$;
the strong and weak ***κ*r- equivalence**: $P\simeq_{\kappa r}Q$ iff $\forall\mathcal{T}.(\mathcal{T}[P]\sim_{\kappa o\tau}\mathcal{T}[Q])$, $P\cong_{\kappa r}Q$ iff $\forall\mathcal{T}.(\mathcal{T}[P]\approx_{\kappa o\tau}\mathcal{T}[Q])$.

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

Definition 5−53: The strong (or weak) *r*1**-bisimulation** is a strong (or weak, respectively) oτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then $[{}^{\dagger}m\langle\tilde{u}\rangle]P\mathcal{S}[{}^{\dagger}m\langle\tilde{u}\rangle]Q$ for all $[{}^{\dagger}m\langle\tilde{u}\rangle]$.

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

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

Lemma 5−54: The responsive equivalence and r1-bisimulation are coincide for both κ-version and non-κ-version, i.e., $\sim_{\kappa r1}\equiv\simeq_{\kappa r}$, $\approx_{\kappa r1}\equiv\cong_{\kappa r}$, $\sim_{r1}\equiv\simeq_r$ and $\approx_{r1}\equiv\cong_r$.

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

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

Definition 5−55 : The (strong) **responsive bisimulation** is a (strong) oτ-bisimulation $\mathcal{S}$ such that whenever $P\mathcal{S}Q$ then $P\xrightarrow{{}^{\dagger}m(\tilde{u})}P'$ implies either $Q\xrightarrow{{}^{\dagger}m(\tilde{u})}Q'$ and $P'\mathcal{S}Q'$, or $Q\xrightarrow{\tau}Q'$ and $P'\mathcal{S}\,[{}^{\dagger}m\langle\tilde{u}\rangle]Q'$.

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

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

Lemma 5−56: The responsive bisimulation and r1-bisimulation are coincide for both κ-version and non-κ-version, i.e., $\sim_{\kappa r}\equiv\sim_{\kappa r1}$, $\approx_{\kappa r}\equiv\approx_{\kappa r1}$, $\sim_r\equiv\sim_{r1}$ and $\approx_r\equiv\approx_{r1}$.

Corollary 5−57: The responsive bisimulation and responsive equivalence are coincide for both κ-version and non-κ-version, i.e., $\sim_{\kappa r}\equiv\simeq_{\kappa r}$, $\approx_{\kappa r}\equiv\cong_{\kappa r}$, $\sim_r\equiv\simeq_r$ and $\approx_r\equiv\cong_r$.

5.2 Properties of the responsive bisimulation

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

Corollary 5−58: The responsive bisimulations are preserved by localisation. That is, let $\mathcal{S}$ be any of $\sim_r$, $\approx_r$, $\sim_{\kappa r}$ or $\approx_{\kappa r}$, then $P\mathcal{S}Q$ implies $[{}^{\dagger}m\langle\tilde{u}\rangle]P\,\mathcal{S}\,[{}^{\dagger}m\langle\tilde{u}\rangle]Q$ for all $[{}^{\dagger}m\langle\tilde{u}\rangle]$.

Lemma 5−59: The responsive bisimulations are equivalences, That is, they are reflexive, symmetric and transitive.

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

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

Definition 5−60: Let ${}^{\dagger}m$ be the input polar of a communication channel name m, P be a process for which $m\in \mathit{fin}(P)$, and $\mathcal{E}$ be the context $\mathcal{E}[.] \overset{\text{def}}{=} (\nu\,\tilde{n})\ (Env\,|\,[.]\,)$ where $m\notin \mathit{fin}(Env)$ while m may or may not be a member of $\tilde{n}$. We say that, P is an *owner* of ${}^{\dagger}m$ (or say, ${}^{\dagger}m$ is owned by P) with respect to the *environment Env*;
Env is an environment free of ${}^{\dagger}m$ (or say, ${}^{\dagger}m$*-free environment*);
$\mathcal{E}[.]$ is an ${}^{\dagger}m$*-safe* environment context, or ${}^{\dagger}m$-safe environment for short.

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

Definition 5−61 A process P is *safe* for *Env*, and the environment *Env* is said to be *safe* for P, if P is the owner of all $m\in \mathit{fin}(P)$ respect to the environment *Env*, i.e., $\mathit{fin}(P)\cap \mathit{fin}(Env)=\varnothing$. 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 $\mathit{fin}(P)=\varnothing$.

Lemma 5−62: The process safety is preserved by evolution. That is, if $\mathit{fin}(P)\cap \mathit{fin}(Env)=\varnothing$ holds for processes P and *Env*, then $\mathit{fin}(P')\cap \mathit{fin}(Env')=\varnothing$ holds for all P' and Env', which are descendants of P and *Env* respectively.
Proof: Simply because the input polar of a channel cannot be transmitted by communication. ■

Corollary 5−63: An autonomous process and all its descendants are safe to any system.

When modelling objects in the κ-calculus, all method bodies can be considered as autonomous, since after parameters passed through the method interface, further input (if any) can only be performed via channels that were initially private and informed to the senders by the forked method body. An object itself is initially autonomous while creation, until its name, the unique identification, is exported to its environment. Its method names can also be considered as initially private to the object, and then exported to the caller during each method call. For example, similar to [Walker95] and

[Zhang97] amongst others, the method call `o.m₁(a₁,a₂)` may be modelled as $(\nu\, mset)(\bar{o}\langle \bar{m}set\rangle \mid \dagger mset(\tilde{m}).\bar{m}_1\langle \bar{a}_1,\bar{a}_2\rangle)$, and on the object side the encoding will look like $(\nu\ \tilde{m})\ (!\dagger o(\bar{m}set).\ \bar{m}set\langle\tilde{m}\rangle \mid \Lambda\circ[\ \bigotimes !\dagger m_i(\tilde{v})L_i.Body_i]\ )$.

Proposition 5−64: The responsive bisimulations are preserved by parallel composition for safe processes. That is, to each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, whenever $P_1\mathcal{S}P_2$ implies $(P_1|P)\mathcal{S}(P_2|P)$ for all P which satisfying $fin(P)\cap(fin(P_1)\cup fin(P_2))=\varnothing$.

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

Proposition 5−65: The responsive bisimulations are preserved by output polarity name substitution. That is, to each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, $P\mathcal{S}Q$ implies $P\sigma\mathcal{S}Q\sigma$ for all $\sigma=\{\tilde{u}/\tilde{x}\}$.

Proposition 5−66: The responsive bisimulations are preserved by restriction. That is, to each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, whenever $P\mathcal{S}Q$ implies $(\nu\,\tilde{n})P\mathcal{S}(\nu\,\tilde{n})Q$ for all $\tilde{n}$.

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

Proposition 5−67: The responsive bisimulations are preserved by *GEC* choice for autonomous processes. That is, to each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, if P_1 and P_2 are autonomous processes, then $P_1\mathcal{S}P_2$ implies $\mathcal{D}[P_1]\mathcal{S}\mathcal{D}[P_2]$ for all process context $\mathcal{D}[.]$ of the form $\mathcal{D}[.]\overset{\text{def}}{=}\Lambda\circ[\,!(\nu\,\kappa)\dagger m(\tilde{x})L.[.]\otimes G]$.

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 5−68: For each κ-version or non-κ-version responsive bisimulation $\mathcal{S}$, if P_1 and P_2 are safe processes, then $P_1\mathcal{S}P_2$ implies $\mathcal{D}[P_1]\mathcal{S}\mathcal{D}[P_2]$ for all process context of the form $\mathcal{D}[.]\overset{\text{def}}{=}\Lambda\circ[\,!\dagger m(\tilde{x})\,(\nu)@J.[.]\otimes G]$ where $m\in J$.

Proposition 5−69: For autonomous processes, the responsive bisimulations are congruences. That is, for each of the κ-version or non-κ-version responsive bisimulations $\mathcal{S}$, if P_1 and P_2 are autonomous processes, then $P_1\mathcal{S}P_2$ implies $\mathcal{C}[P_1]\mathcal{S}\mathcal{C}[P_2]$ for all process context $\mathcal{C}[.]$.

5.3 Coincidence between some variations of responsive bisimulation

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

Definition 5−70: Each of the following variations of responsive bisimulations is a (strong) οτ-bisimulation $\mathcal{S}$:

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

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

The (strong) **open responsive bisimulation** is a symmetric relation $\mathcal{S}$ on processes if whenever $P\mathcal{S}Q$ then for any output communication polar substitution $\sigma=\{\tilde{y}/\tilde{x}\}$, we have

$P\sigma\xrightarrow{\alpha}P'$ implies $\exists Q'$ s.t. $Q\sigma\xrightarrow{\alpha}Q'$ and $P'\mathcal{S}Q'$, where either $\alpha=(\nu\,\tilde{v})\bar{m}\langle\tilde{u}\rangle$ and $\tilde{v}\cap fn(Q)=\emptyset$, or $\alpha=\tau$;
$P\xrightarrow{^{+}m(\tilde{u})}P'$ implies $\exists Q'$ s.t. either $Q\sigma\xrightarrow{^{+}m(\tilde{u})}Q'$ and $P'\mathcal{S}Q'$ or $Q\sigma\xrightarrow{\tau}Q'$ and $P'\mathcal{S}[^{+}m\langle\tilde{u}\rangle]Q'$.

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

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

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

Lemma 5−71: The early, late and open responsive bisimulations all coincide with the standard version of responsive bisimulation.

5.4 Conventional bisimulations

Most familiar bisimulation relations which are widely used in convensional π-calculus can be also defined in the κ-calculus, with the similar style as we did for the polar π-calculus ([Zhang02A]).

Definition 5−72: The (strong) **ground bisimulation** is a (strong) οτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then
$P\xrightarrow{^{+}m(\tilde{u})}P'$ implies either $Q\xrightarrow{^{+}m(\tilde{u})}Q'$.

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

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

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

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

Lemma 5−73: The ground bisimulation, early bisimulation, late bisimulation and open bisimulation are all coincided in the κ-calculus.
Proof: First, the ground bisimulations are preserved by output polarity name substitution, this can be proven in a way similar to that for Proposition 5-65, except no need to check the cases involving localisation, then the lemma is followed. ■

Corollary 5−74: The ground bisimulations are responsive bisimulations, that is, $\sim_{g}\subseteq\sim_{r}$ and $\approx_{g}\subseteq\approx_{r}$.
Proof: Directly concluded from the comparison of their definitions. ■

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

Definition 5-75: The (strong) **asynchronous bisimulation** is a (strong) oτ-bisimulation $\mathcal{S}$ if whenever $P\mathcal{S}Q$ then $P\xrightarrow{\overline{m}(\tilde{u})}P'$ implies either $Q\xrightarrow{\overline{m}(\tilde{u})}Q'$ and $P'\sim_a Q'$, or $Q\xrightarrow{\tau}Q'$, and $P'\mathcal{S}(\overline{m}\langle\tilde{u}\rangle\,|\,Q')$.

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

As pointed out in [Zhang02A], 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 5-4. The asynchronous bisimulation is not interested in because the following reasons:

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

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

Definition 5-76: We say that all the responsive bisimulation, asynchronous bisimulation, ground bisimulation, early bisimulation, late bisimulation and open bisimulation are **input-related bisimulations**.

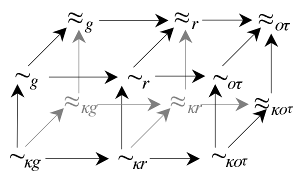

Figure 5-3

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

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

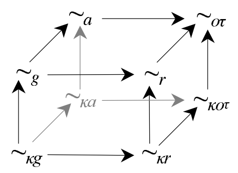

Figure 5-4

The Figure 5-3 and Figure 5-4 summarise some bisimulation relations discussed so far, from the strongest one, $\sim_{\kappa g}$, to the weakest, $\approx_{o\tau}$, where each arrow respresnets a "$\subseteq$", or "is a subset of", relation.

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

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

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

6 Choice Equivelance

Above properties are based on the view at processes level, and do not give us much room to describe what happen inside a GEC choice. To restrict a view within the scope of a GEC term, we introduce the transition of GEC choices:

Definition 6−77: The GEC choice term G can commit on input polar $\overset{+}{m}\in{}^{+}\mathcal{M}$, denoted as $G\downarrow\overset{+}{m}$, if $(\lfloor\rfloor\circ[G])\downarrow\overset{+}{m}$.
The GEC choice term G can commit the input action $\alpha=\overset{+}{m}(\tilde{u})$, denoted as $G\downarrow\alpha$, if $(\lfloor\rfloor\circ[G])\downarrow\alpha$.

Lemma 6−78: All possible transition a GEC choice may take can be described by:

$(\Lambda\circ[G])\downarrow\overset{+}{m}$ iff $\overset{+}{m}\in guard(G)$ and $\Lambda\downarrow\overset{+}{m}$; $(\lfloor\rfloor\circ[G])\not\downarrow\check{\kappa}$ for all κ; $(\Lambda\circ[G])\downarrow\check{\kappa}$ iff $\Lambda\downarrow\check{\kappa}@guard(G)$.

Proof: Apply Definition-3-23, Figure 3-1 and Definition 4-43 to the reduction rules listed in Figure 4-2, then concluded by induction. ■

Corollary 6−79: Let $\mathcal{S}$ be one of the input-related bisimulations, if $\Lambda_1\circ[G_1]\,\mathcal{S}\,\Lambda_2\circ[G_2]$ then $guard(G_1)\equiv guard(G_2)$ and $sig(\Lambda_1,guard(G_1))\equiv sig(\Lambda_2,guard(G_2))$.

Definition 6−80 (*G*-**bisimulation**): Let $\mathcal{S}$ be one of the input-related bisimulations, then $\mathcal{S}_G$, the *G*-bisimulation w.r.t. $\mathcal{S}$, is the largest symmetric relation $\mathcal{R}_G$ on choice terms such that $\mathcal{R}_G\subseteq\mathcal{S}_G$ iff whenever $G_1\mathcal{R}_G G_2$ then $\lfloor\rfloor\circ[G_1]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_1]\,|\,P_1)$ implies $\lfloor\rfloor\circ[G_2]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_2]\,|\,P_2)$, and $P_1\mathcal{S}P_2$.

Lemma 6−81: $\mathcal{S}_G$ are equivalences.

Proof: Reflexive: $G\mathcal{S}_G G$ for any G, according to the definition of $\mathcal{S}_G$;

Symmetric: if $G_1\mathcal{S}_G G_2$ then $G_2\mathcal{S}_G G_1$, by the definition of $\mathcal{S}_G$;

Transitive: Let $G_1\mathcal{R}_1G_2$ and $G_2\mathcal{R}_2G_3$, where $\mathcal{R}_1\subseteq\mathcal{S}_G$ and $\mathcal{R}_2\subseteq\mathcal{S}_G$, and therefore $G_1(\mathcal{R}_1\mathcal{R}_2)G_3$. Assume $\lfloor\rfloor\circ[G_1]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_1]\,|\,P_1)$, then by $G_1\mathcal{R}_1G_2$, it implies $\lfloor\rfloor\circ[G_2]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_2]\,|\,P_2)$ and $P_1\mathcal{S}P_2$. By $G_2\mathcal{R}_2G_3$, it further implies $\lfloor\rfloor\circ[G_3]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_3]\,|\,P_3)$ and $P_2\mathcal{S}P_3$. Since $\mathcal{S}$ is transitive, $P_1\mathcal{S}P_3$, by the definition, $(\mathcal{R}_1\mathcal{R}_2)\subseteq\mathcal{S}_G$. ■

Lemma 6−82: If $G_1\mathcal{S}_G G_2$ then $guard(G_1)\equiv guard(G_2)$ and $excl(G_1)\equiv excl(G_2)$.

Proof: Assume the contraries, $guard(G_1)\not\equiv guard(G_1)$ or $excl(G_1)\not\equiv excl(G_2)$, then $G_1\mathcal{S}_G G_2$ will fail for any $\mathcal{S}$. ■

Lemma 6−83: If $G_1\mathcal{S}_G G_2$ then $(G_1\otimes G)\mathcal{S}_G(G_2\otimes G)$ for any G.

Proof: Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the relations between $G_1\otimes G$ and $G_2\otimes G$, such that $G_1\otimes G\,\mathcal{R}_1\,G_2\otimes G$ and $G_2\otimes G\,\mathcal{R}_2\,G_1\otimes G$, then we may have the symmetric relation $\mathcal{R}=\mathcal{R}_1\cup\mathcal{R}_2$. For $\mathcal{R}_1$ we assume $(G_1\otimes G)\downarrow\overset{+}{m}$, then it must be either

$\lfloor\rfloor\circ[G_1]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_1]\,|\,P_1)$ and $\lfloor\rfloor\circ[G_1\otimes G]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_1\otimes G]\,|\,P_1)$ or

$\lfloor\rfloor\circ[G]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G]\,|\,P)$ and $\lfloor\rfloor\circ[G_1\otimes G]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_1\otimes G]\,|\,P)$.

For the former, by $G_1\mathcal{S}_G G_2$ it implies $\lfloor\rfloor\circ[G_2]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_2]\,|\,P_2)$ and $\lfloor\rfloor\circ[G_2\otimes G]\xrightarrow{\overset{+}{m}(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_2\otimes G]\,|\,P_2)$ and $P_1\mathcal{S}P_2$.

For the latter we simply have $\lfloor\rfloor\circ[G_2\otimes G]\xrightarrow{{\dagger}m(\tilde{u})}(\nu\kappa)(\Lambda\circ[G_2\otimes G]\,|\,P)$ and certainly $P\mathcal{S}P$.

By assume $(G_2\otimes G)\downarrow{\dagger}m$ for $\mathcal{R}_2$, we got the symmetric result, therefore by definition of $\mathcal{S}_G$, we have $\mathcal{R}\subseteq\mathcal{S}_G$. ■

Lemma 6−84: If $G_1\mathcal{S}_G G_2$ and $G_3\mathcal{S}_G G_4$, then $G_1\otimes G_3\mathcal{S}_G G_2\otimes G_4$.
Proof: Since $G_1\mathcal{S}_G G_2$ and $G_3\mathcal{S}_G G_4$, by Lemma-6-83, $G_1\otimes G_3\mathcal{S}_G G_2\otimes G_3$ and $G_3\otimes G_2\mathcal{S}_G G_4\otimes G_2$. By rule str_SUM2, $G_2\otimes G_3\mathcal{S}_G G_2\otimes G_4$. Apply Lemma-6-81 to $G_1\otimes G_3\mathcal{S}_G G_2\otimes G_3$ and $G_2\otimes G_3\mathcal{S}_G G_2\otimes G_4$, therefore $G_1\otimes G_3\mathcal{S}_G G_2\otimes G_4$. ■

Corollary 6−85: From the definition of $\mathcal{S}_G$, the following properties can easliey be concluded.
1. $(\nu\,\kappa)!{\dagger}m(\tilde{x})\check{\kappa}@J.P\ \mathcal{S}_G\ !(\nu\,\kappa){\dagger}m(\tilde{x})\check{\kappa}@J.P$, for either ${\dagger}m\in J$ or $\hat{\kappa}\notin fon(P)$;
2. $\bigotimes_{i\in I}(\nu\,\kappa)!{\dagger}m_i(\tilde{x})\check{\kappa}@J_i.P_i\ \mathcal{S}_G\ (\nu\,\kappa)\bigotimes_{i\in I}!{\dagger}m_i(\tilde{x})\check{\kappa}@J_i.P_i$, if $\hat{\kappa}\notin fon(P_i)$;
3. $G\otimes\ldots\otimes G\ \mathcal{S}_G\ G$, where $G\otimes\ldots\otimes G$ involves finite copies of G.

Proof: The proofs for 1st and 2nd clauses are trivial, can be derived directly for the definition of the G-bisimulation.

The 3rd clause can be proven by induction: $G\mathcal{S}_G G$ has be given in Lemma-6-81; $G\otimes G\ \mathcal{S}_G\ G$ can be trivially obtained from the definition of the G-bisimulation; assume $(\bigotimes_k G)\mathcal{S}_G G$, then by Lemma-6-83 we have $(\bigotimes_k G)\otimes G\ \mathcal{S}_G\ G\otimes G$, and by Lemma-6-81 and $G\otimes G\ \mathcal{S}_G\ G$, we have $(\bigotimes_{k+1}G)\mathcal{S}_G G$. ■

Lemma 6−86: Whenever $\mathcal{S}$ can be preserved by parallel composition, then $G_1\mathcal{S}_G G_2$ implies $\Lambda\circ[G_1]\,\mathcal{S}\,\Lambda\circ[G_2]$ for all Λ.
Proof: Assume an arbitrary locking list Λ, and in the domain where $\mathcal{S}$ can be preserved by parallel composition, assume two arbitrary processes P_1 and P_2 which satisfy $P_1\mathcal{S}P_2$. Let $\mathcal{R}$ be a symmetric relation where $(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1)\mathcal{R}(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2)$. We first prove that $G_1\mathcal{S}_G G_2$ and $P_1\mathcal{S}P_2$ implies $\mathcal{R}\subseteq\mathcal{S}$. Since $\mathbf{0_P}\mathcal{S}\,\mathbf{0_P}$, this lemma becomes a special case where $\tilde{n}$ is an empty set and $P_1\equiv P_2\equiv\mathbf{0_P}$.

Here we only show that for $\mathcal{S}\subseteq\sim_r$. That for other input-related bisimulations can be proven similarly.

First, by the definition of guarded choice terms, both $\Lambda\circ[G_1]$ and $\Lambda\circ[G_2]$ may commit on input actions but cannot commit on any output action, neither $\overline{m}\langle\tilde{u}\rangle$ nor $\check{\kappa}$, nor any internal action τ.

Second, to meet the condition of $\sim_r$ being preserved by parallel composition, we restrict that all processes concerned in this proof are with the safe process domain.

By Lemma-6-82 $guard(G_1)\equiv guard(G_2)$, therefore $\Lambda\circ[G_1]\xrightarrow{\check{\kappa}}\Lambda'\circ[G_1]$ implies $\Lambda\circ[G_2]\xrightarrow{\check{\kappa}}\Lambda'\circ[G_2]$, and vice versa. It is clearly $(\nu\,\tilde{n})(\Lambda'\circ[G_1]\,|\,P_1)\mathcal{R}(\nu\,\tilde{n})(\Lambda'\circ[G_2]\,|\,P_2)$.

Assume action $\alpha={\dagger}m(\tilde{u})$ and $(\Lambda\circ[G_1])\downarrow\alpha$, then it must $\Lambda\downarrow{\dagger}m$ and $G_1\downarrow{\dagger}m$. Let

$\lfloor\rfloor\circ[G_1]\xrightarrow{\alpha}(\nu\kappa)(\Lambda_1\circ[G_1]\,|\,Q_1)$, by $G_1\mathcal{S}_G G_2$ it implies $\lfloor\rfloor\circ[G_2]\xrightarrow{\alpha}(\nu\kappa)(\Lambda_1\circ[G_2]\,|\,Q_2)$ and $Q_1\mathcal{S}Q_2$.

It also implies there is a lock L rised by $G_1\downarrow{\dagger}m$ such that $\lfloor\rfloor\xrightarrow{{\dagger}m\uparrow L}\Lambda_1$, and $\Lambda\xrightarrow{{\dagger}m\uparrow L}\Lambda'$. Therefore we have

$\Lambda\circ[G_1]\xrightarrow{\alpha}(\nu\kappa)(\Lambda'\circ[G_1]\,|\,Q_1)$ and $\Lambda\circ[G_2]\xrightarrow{\alpha}(\nu\kappa)(\Lambda'\circ[G_2]\,|\,Q_2)$.

Let $P_1'\overset{\text{def}}{\equiv}Q_1\,|\,P_1$ and $P_2'\overset{\text{def}}{\equiv}Q_2\,|\,P_2$, then within the safe process domain we have $P_1'\mathcal{S}P_2'$. By rule tr_PARL and str-SCP4,

$(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1)\xrightarrow{\alpha}(\nu\,\tilde{n},\kappa)(\Lambda'\circ[G_1]\,|\,P_1')$, $(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_1)\xrightarrow{\alpha}(\nu\,\tilde{n},\kappa)(\Lambda'\circ[G_2]\,|\,P_2')$

and therefore $(\nu\,\tilde{n},\kappa)(\Lambda'\circ[G_1]\,|\,P_1')\,\mathcal{R}\,(\nu\,\tilde{n},\kappa)(\Lambda'\circ[G_2]\,|\,P_2')$.

Assume $P_1\xrightarrow{{\dagger}m(\tilde{u})}P_1'$, then $(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1)\xrightarrow{{\dagger}m(\tilde{u})}(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1')$, and by $P_1\mathcal{S}P_2$ we have either $P_2\xrightarrow{{\dagger}m(\tilde{u})}P_2'$ and $P_1'\mathcal{S}P_2'$, or $P_2\xrightarrow{\tau}P_2'$ and $P_1'\,\mathcal{S}\,[{\dagger}m\langle\tilde{u}\rangle]P_2'$.

For the former we have $(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2)\xrightarrow{{\dagger}m(\tilde{u})}(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2')$, and therefore $(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_2')\mathcal{R}(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2')$. For the latter, within the safe process domain we have $(\Lambda\circ[G_2]\,|\,[{\dagger}m\langle\tilde{u}\rangle]P_2')\equiv[{\dagger}m\langle\tilde{u}\rangle](\Lambda\circ[G_2]\,|\,P_2')$, therefore we have $(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2)\xrightarrow{\tau}(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2')$ and $(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1')\mathcal{R}[{\dagger}m\langle\tilde{u}\rangle]\,(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2'))$.

Assume arbitrary other kind action α for which $P_1\xrightarrow{\alpha}P_1'$, then by $P_1\mathcal{S}P_2$, we have $P_2\xrightarrow{\alpha}P_2'$, $P_1'\mathcal{S}P_2'$, then

$(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1)\xrightarrow{\alpha}(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1')$, $(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2)\xrightarrow{\alpha}(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2')$, and therefore $(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1')\mathcal{R}(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2')$.

Assume $\alpha=\tau$, which is caused by $\Lambda\circ[G_1]\downarrow\check{\kappa}$ and $P_1\downarrow\hat{\kappa}$, then from above it is easy to see

$$(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1)\xrightarrow{\tau}(\nu\,\tilde{n})(\Lambda'\circ[G_1]\,|\,P_1'),\qquad(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2)\xrightarrow{\tau}(\nu\,\tilde{n})(\Lambda'\circ[G_2]\,|\,P_2'),$$

and therefore $(\nu\,\tilde{n})(\Lambda'\circ[G_1]\,|\,P_1')\mathcal{R}(\nu\,\tilde{n})(\Lambda'\circ[G_2]\,|\,P_2')$.

Assume $\alpha=\tau$, which is caused by $\Lambda\circ[G_1]\downarrow\dagger m(\tilde{u})$ and $P_1\downarrow(\nu\,\tilde{v})\bar{m}\langle\tilde{u}\rangle$, where $\tilde{v}\subseteq\tilde{u}$, then from above it is easy to see

$$(\nu\,\tilde{n})(\Lambda\circ[G_1]\,|\,P_1)\xrightarrow{\tau}(\nu\,\tilde{n})(\nu\,\tilde{v})(\Lambda'\circ[G_1]\,|\,P_1'),\qquad(\nu\,\tilde{n})(\Lambda\circ[G_2]\,|\,P_2)\xrightarrow{\tau}(\nu\,\tilde{n})(\nu\,\tilde{v})(\Lambda'\circ[G_2]\,|\,P_2'),$$

and therefore $(\nu\,\tilde{n},\tilde{v})(\Lambda'\circ[G_1]\,|\,P_1')\mathcal{R}(\nu\,\tilde{n},\tilde{v})(\Lambda'\circ[G_2]\,|\,P_2')$.

Since all possible transitions are covered in above cases, by the definition of $\mathcal{S}$ (here it is $\sim_r$), we have $\mathcal{R}\subseteq\mathcal{S}$. ■

Lemma 6-87: In the domain where $\mathcal{S}$ can be preserved by parallel composition, if $G_1\mathcal{S}_G G_2$, $G_3\mathcal{S}_G G_4$, $G_5\mathcal{S}_G G_6$ and $G_7\mathcal{S}_G G_8$ then $\Lambda\circ[G_1\otimes G_3]\,|\,\Lambda'\circ[G_5\otimes G_7]\mathcal{S}\,\Lambda\circ[G_2\otimes G_4]\,|\,\Lambda'\circ[G_6\otimes G_8]$.

Proof: By Lemma-6-84 and Lemma-6-86, $\Lambda\circ[G_1\otimes G_3]\mathcal{S}\ \Lambda\circ[G_2\otimes G_4]$ and $\Lambda'\circ[G_5\otimes G_7]\mathcal{S}\Lambda'\circ[G_6\otimes G_8]$. But $\mathcal{S}$ is preserved by parallel composition. ■

In section 11, we will point it out that the G-bisimulation indicating the behaviours simularity between objects.

7 Higher Order Extension

Here is a problem of the *G*-bisimulations: they are too strong, they have to depend on process bisimulation, and cannot always percisely describe the behave equivalence in a finer grain, ie., on individual choice term, when involving parallel composition of GEC choices. For example, lets look at the following processes O_1 and O_2:

$$\begin{array}{ll} O_1\overset{\text{def}}{=}(\nu\ n_a,n_b)\,(\Lambda\circ[G_1]\,|\,\lfloor\,\rfloor\circ[G_1']\,), & G_1\overset{\text{def}}{=}!(\nu\,\kappa)\dagger m_a(\tilde{x})L_a.\bar{n}_a\langle\tilde{x},\kappa\rangle\otimes\,!(\nu\,\kappa_b)\dagger m_b(\tilde{x})L_b.\bar{n}_b\langle\tilde{x},\kappa\rangle, \\ & G_1'\overset{\text{def}}{=}!\dagger n_a(\tilde{x},\kappa).P_a\otimes\,!\dagger n_b(\tilde{x},\kappa).P_b \\ O_2\overset{\text{def}}{=}\Lambda\circ[G_2], & G_2\overset{\text{def}}{=}!(\nu\,\kappa)\dagger m_a(\tilde{x})L_a.P_a\otimes\,!(\nu\,\kappa_b)\dagger m_b(\tilde{x})L_b.P_b \end{array}\qquad 7\text{-}1$$

though $O_1\approx_g O_2$ is easy observed, it is difficult to describe similarities in choice term level, since neither $G_1\approx_{gG}G_2$ nor $G_1'\approx_{gG}G_2$ is held. To solve the problem, we need something else to capture the *GEC* structural characteristics of G_1 and G_2 only, without involve with the behaviours of P_a and P_b.

Consider other two processes, Q_1 and Q_2:

$$\begin{array}{l} Q_1\overset{\text{def}}{=}(\nu\,\kappa_1)(\lfloor\check{\kappa}_1@[\dagger m_b]\rfloor\circ[!(\nu\,\kappa)\dagger m_a(\tilde{x})\check{\kappa}@[\dagger m_b].P_a\otimes\,!(\nu\,\kappa)\dagger m_b(\tilde{x})\check{\kappa}@[\dagger m_a].P_b]\,|\,\hat{\kappa}_1\,) \\ Q_2\overset{\text{def}}{=}(\nu\,\kappa_2)(\lfloor\check{\kappa}_2@[\dagger m_b]\rfloor\circ[!(\nu\,\kappa)\dagger m_a(\tilde{x})\check{\kappa}@[\dagger m_b].P_a\otimes\,!(\nu\,\kappa)\dagger m_b(\tilde{x})\check{\kappa}@[\dagger m_a].P_a\otimes\,!(\nu\,\kappa)\dagger m_b(\tilde{x})\check{\kappa}@[\dagger m_a].P_b]\,|\,\hat{\kappa}_2\,) \end{array}\qquad 7\text{-}2$$

Provide $\hat{\kappa}_1,\hat{\kappa}_2\notin(\mathit{fon}(P_a)\cup\mathit{fon}(P_b))$, then it is clear that Q_1 and Q_2 have the same behaviour, regardless what P_a and P_b are. If we substitute P_a with P_c, and P_b with P_d in both processes Q_1 and Q_2, then these two will still be equivalent to each other, as long as $\hat{\kappa}_1,\hat{\kappa}_2\notin(\mathit{fon}(P_c)\cup\mathit{fon}(P_d))$. That means that the behaviours of Q_1 and Q_2 have some common features independent from P_a and P_b. To capture these kinds of behaviour features, we introduce the notion of *higher order* process into the algebra to abstract P_a, P_b away. For example, we may rewrite Q_1 and Q_2 in the following form:

$$\begin{array}{l} Q_1\overset{\text{def}}{=}\mathbb{Q}_1«P_a,P_b»\langle\dagger m_a,\dagger m_b\rangle,\quad Q_2\overset{\text{def}}{=}\mathbb{Q}_2«P_a,P_b»\langle\dagger m_a,\dagger m_b\rangle,\qquad\text{where} \\ \mathbb{Q}_1«\eta_1,\eta_2»(\dagger y,\dagger z)\overset{\text{def}}{=}(\nu\,\kappa_1)((\lfloor\check{\kappa}_1@[\dagger z]\rfloor\circ[!(\nu\,\kappa)\dagger y(\tilde{x})\check{\kappa}@[\dagger z].\eta_1\otimes\,!(\nu\,\kappa)\dagger z(\tilde{x})\check{\kappa}@[\dagger y].\eta_2]\,|\,\hat{\kappa}_1\,) \\ \mathbb{Q}_2«\eta_1,\eta_2»(\dagger y,\dagger z)\overset{\text{def}}{=}(\nu\,\kappa_2)((\lfloor\check{\kappa}_2@[\dagger z]\rfloor\circ[!(\nu\,\kappa)\dagger y(\tilde{x})\check{\kappa}@[\dagger z].\eta_1\otimes\,!(\nu\,\kappa)\dagger z(\tilde{x})\check{\kappa}@[\dagger y].\eta_2\otimes\,!(\nu\,\kappa)\dagger z(\tilde{x})\check{\kappa}@[\dagger y].\eta_2]\,|\,\hat{\kappa}_2) \end{array}$$

Here $\mathbb{Q}_1$ and $\mathbb{Q}_2$ are higher (2nd) order processes which take processes as arguments, and η_1 and η_2, the parameters within the brackets « », are process variables.

In order to deal with higher order terms separately, we may change the writing style further into the form $Q_1 \equiv \mathbb{Q}_1\langle \overline{m}_a, \overline{m}_b\rangle «(P_a, P_b)$, and call the tuple (P_a, P_b) the *continuation* of the function $\mathbb{Q}_1\langle \overline{m}_a, \overline{m}_b\rangle$.

Close to those in [Liu97], [Philippou96], [Zhang98A] and [Zhang98B], the higher-order extension to the process algebra in this paper is only involved with higher-order process abstractions but excludes higher-order communication ([Sangiorgi92a], [Sangiorgi92b]), and therefore allows the calculus to employ the relatively simpler bisimilarity theory of the π–calculus. The major significant of this higher-order extension is its ability to separate the exclusion behaviours from other behaviours, and the introduction of $\mathbb{G}$-bisimilation.

7.1 Extension in syntax and semantic

Extended syntax: Let $\mathbb{P}$ be higher-order process given by $\mathbb{P} \overset{\text{def}}{=} «\tilde{\eta}»P$, $\mathbb{G}$ be higher-order GEC given by $\mathbb{G} \overset{\text{def}}{=} «\tilde{\eta}»\mathbb{H}$, η be process variable, $\mathbb{H}$ the higher-order GEC terms, Б be a branches of a higher-order GEC terms, etc. The using of outlined fonts for those symbol is to implicitly remind us that they process variables to be filled with.

$$\mathbb{H} ::= \mathbf{0_G} \mid \mathbb{H}_1 \otimes \mathbb{H}_2 \mid (\nu\tilde{n})\mathbb{H} \mid \mathbb{G}«\tilde{\eta}», \qquad \text{Б} ::= !\beta.\eta \mid (\nu\,\kappa)\,!\beta.\eta \mid !(\nu\,\kappa)\beta.\eta$$

Let $\tilde{P}$ be the abbreviation of $\{P_{i\in I}\}$, then syntax of process terms P and GEC choice terms G can be re-defined as

$$P ::= \mathbf{0_P} \mid \overline{m}\langle\tilde{u}\rangle \mid \hat{\kappa} \mid (\nu\,\tilde{n})P \mid P_1 \,|\, P_2 \mid \Lambda\circ[G] \mid A\langle\tilde{a}\rangle \mid \mathbb{P}«\tilde{P}» \mid \eta$$
$$G ::= B \mid (\nu\,\tilde{n})G \mid G_1 \otimes G_2 \mid D\langle\tilde{a}\rangle \mid \mathbb{G}«\tilde{P}»$$

Careful readers may have noticed that P is allowed to be recursively defined through $\mathbb{P}$, but G can not be recursively defined through $\mathbb{G}$, and there is no "*GEC* variable". This is because the ability to separate the exclusive behaviours will be otherwise lost. $(\tilde{w})«\tilde{\eta}»P$ is a higher-order process *abstraction*, where $\tilde{\eta}$ contain all the freely occurred process variables in P, correspondingly, $«\tilde{\eta}»\mathbb{G}$ is a higher-order GEC *abstraction*.

We may consider $\mathbb{P}$ and $\mathbb{G}$ are $\tilde{P} \mapsto P$ and $\tilde{P} \mapsto G$ functions respectively. The semantics of the higher-order terms are described by the rules in the following table:

str_HARG: $(«\tilde{\eta}»\mathbb{H})«\tilde{P}» \equiv \mathbb{H}\{\tilde{P}/\tilde{\eta}\}$	str_HARP: $(«\tilde{\eta}»\mathbb{P})«\tilde{P}» \equiv \mathbb{P}\{\tilde{P}/\tilde{\eta}\}$

Definition 7−88: Higher-order process terms $\mathbb{P}_1$ and $\mathbb{P}_2$ are structural equivalent, written as $\mathbb{P}_1 \equiv \mathbb{P}_2$ if $arity(\mathbb{P}_1) = arity(\mathbb{P}_2)$, and $\mathbb{P}_1«\tilde{P}» \equiv \mathbb{P}_2«\tilde{P}»$ for any $\tilde{P}$ satisfying $arity(\tilde{P}) = arity(\mathbb{P}_1)$.

Higher-order GEC terms $\mathbb{G}_1$ and $\mathbb{G}_2$ are structural equivalent, written $\mathbb{G}_1 \equiv \mathbb{G}_2$ if $arity(\mathbb{G}_1) = arity(\mathbb{G}_2)$, and $\mathbb{G}_1«\tilde{P}» \equiv \mathbb{G}_2«\tilde{P}»$ for any $\tilde{P}$ satisfying $arity(\tilde{P}) = arity(\mathbb{G}_1)$.

For example, $\mathbb{G}_1 \equiv \mathbb{G}_2$ if $\mathbb{G}_1 \overset{\text{def}}{=} «\eta_1,\eta_2»(!\beta_1.\eta_1 \otimes !\beta_2.\eta_2)$ and $\mathbb{G}_2 \overset{\text{def}}{=} «\eta_1,\eta_2»(!\beta_2.\eta_2 \otimes !\beta_1.\eta_1)$.

Lemma 7−89: If $arity(\mathbb{P}_1) = arity(\mathbb{P}_2)$ and $\mathbb{P}_1 \equiv «\tilde{\eta}»\mathbb{P}_2«\tilde{\eta}»$ then $\mathbb{P}_1 \equiv \mathbb{P}_2$.
If $arity(\mathbb{G}_1) = arity(\mathbb{G}_2)$ and $\mathbb{G}_1 \equiv «\tilde{\eta}»\mathbb{G}_2«\tilde{\eta}»$ then $\mathbb{G}_1 \equiv \mathbb{G}_2$.

Proof: For an arbitrary $\tilde{P}$ satisfying $arity(\tilde{P}) = arity(\mathbb{P}_1)$, we have $\mathbb{P}_1«\tilde{P}» \equiv \mathbb{P}_2«\tilde{P}»$. Similar we have $\mathbb{G}_1«\tilde{P}» \equiv \mathbb{G}_2«\tilde{P}»$. ■

Corollary 7−90: $\mathbb{G} \equiv «\tilde{\eta}»\mathbb{G}«\tilde{\eta}»$.

Notation 7−91: For convenience, we sometimes using symbol $\mathbb{B}$ to denote a higher-order GEC $\mathbb{G}$ with single arity and single branch, that is, it has the form of $\mathbb{B} \overset{\text{def}}{=} «\eta»\text{Б}$.

With a slight notation abuse, whenever these is no ambiguity, we may simply write $\mathbb{G}«\tilde{\eta}»$ as $\mathbb{G}$ (and therefore write $\mathbb{B}«\eta»$ as $\mathbb{B}$), then symbol $\mathbb{H}$ is no-longer necessary.

Definition 7−92 ($\mathbb{P}$ **−equivalence**): Given a process bisimulation $\mathcal{S}$, then the symmetric relation $\mathcal{S}_{\mathbb{P}}$ on higher order process terms is a $\mathbb{P}$−bisimulation with respect to $\mathcal{S}$, iff whenever $\mathbb{P}_1\ \mathcal{S}_{\mathbb{P}}\ \mathbb{P}_2$ then $arity(\mathbb{P}_1)=arity(\mathbb{P}_2)$, and $(\mathbb{P}_1«\tilde{P}»)\mathcal{S}(\mathbb{P}_2«\tilde{P}»)$ is held for all $\tilde{P}$ satisfy $arity(\tilde{P})=arity(\mathbb{P}_1)$.

Lemma 7−93: $(\mathbb{P}_1«\tilde{P}»)\mathcal{S}_{\mathbb{P}}(\mathbb{P}_2«\tilde{P}»)$ whenever $\mathbb{P}_1\mathcal{S}_{\mathbb{P}}\mathbb{P}_2$ and $arity(\tilde{P})\leq arity(\mathbb{P}_1)$.
Proof: Assume some process variables $\tilde{\eta}$ and $\tilde{\eta}'$ such that $arity(\tilde{\eta})=arity(\tilde{P})$ and $arity(\tilde{\eta},\tilde{\eta}')=arity(\mathbb{P}_1)$, then $\mathbb{P}_1=(«\tilde{\eta},\tilde{\eta}'»\mathbb{P}_1)«\tilde{\eta},\tilde{\eta}'»$ and $\mathbb{P}_2=(«\tilde{\eta},\tilde{\eta}'»\mathbb{P}_2)«\tilde{\eta},\tilde{\eta}'»$, therefore $\mathbb{P}_1«\tilde{P}»=(«\tilde{\eta}'»\mathbb{P}_1«\tilde{P}»)«\tilde{\eta}'»$ and $\mathbb{P}_2«\tilde{P}»=(«\tilde{\eta}'»\mathbb{P}_2«\tilde{P}»)«\tilde{\eta}'»$. Let $\mathbb{P}_1'\overset{\text{def}}{=}\mathbb{P}_1«\tilde{P}»$, $\mathbb{P}_2'\overset{\text{def}}{=}\mathbb{P}_2«\tilde{P}»$, then $arity(\mathbb{P}_1')=arity(\mathbb{P}_2')=arity(\tilde{\eta}')$. Since for all $\tilde{Q}$ satisfy $arity(\tilde{Q})=arity(\mathbb{P}_1')$, $\mathbb{P}_1'«\tilde{Q}»=\mathbb{P}_1«\tilde{P},\tilde{Q}»$ and $\mathbb{P}_2'«\tilde{Q}»=\mathbb{P}_2«\tilde{P},\tilde{Q}»$, therefore $(\mathbb{P}_1'«\tilde{Q}»)\ \mathcal{S}\ (\mathbb{P}_2'«\tilde{Q}»)$. ■

Definition 7−94 (**labelled transition of** $\mathbb{P}$): Let $\mathbb{P}$ be be higher-order process, α be an action, we say that $\mathbb{P}$ can take the action α and reduce to $\mathbb{P}'$, denoted as $\mathbb{P}\xrightarrow{\alpha}\mathbb{P}'$, if there exists some higher-order process $\mathbb{P}'$ such that $arity(\mathbb{P}')=arity(\mathbb{P})$, and $\mathbb{P}«\tilde{P}»\xrightarrow{\alpha}\mathbb{P}'«\tilde{P}»$ holds for all $\tilde{P}$ satisfying $arity(\tilde{P})=arity(\mathbb{P})$.
Definition 7−95: Let $arity(\tilde{\eta}_1)=arity(\mathbb{G}_1)$, $arity(\tilde{\eta}_2)=arity(\mathbb{G}_2)$ and $\tilde{\eta}_1\cap\tilde{\eta}_2=\varnothing$, then the choice composition of $\mathbb{G}$ functions is defined as

$$\mathbb{G}_1\otimes\mathbb{G}_2\overset{\text{def}}{=}«\tilde{\eta}_1,\tilde{\eta}_2»\,(\,\mathbb{G}_1«\tilde{\eta}_1»\otimes\mathbb{G}_2«\tilde{\eta}_2»\,).$$

Note that $\mathbb{G}_1\otimes\mathbb{G}_2\not\equiv\mathbb{G}_2\otimes\mathbb{G}_1$, because $(\mathbb{G}_1\otimes\mathbb{G}_2)«\tilde{P}_1,\tilde{P}_2»\not\equiv(\mathbb{G}_2\otimes\mathbb{G}_1)«\tilde{P}_1,\tilde{P}_2»$. This is a disadvantage of the current form of the κ-calculus, since it does not reflect the symmetric beauty of $G_1\otimes G_2\equiv G_2\otimes G_1$. Though this can be fixed by introducing labelled parameters to $\mathbb{G}$, in this moment we do not do so for avoiding to complicate the syntax and rules of the higher-order κ-calculus. However, with the current form we still have $(\mathbb{G}_1\otimes\mathbb{G}_2)«\tilde{P}_1,\tilde{P}_2»\equiv(\mathbb{G}_2\otimes\mathbb{G}_1)«\tilde{P}_2,\tilde{P}_1»$, which structurally reflects $\mathbb{G}_1«\tilde{P}_1»\otimes\mathbb{G}_2«\tilde{P}_2»\equiv\mathbb{G}_2«\tilde{P}_2»\otimes\mathbb{G}_1«\tilde{P}_1»$.

Lemma 7−96: $(\mathbb{G}_1\otimes\mathbb{G}_2)\,«\tilde{P}_1,\tilde{P}_2»\equiv(\mathbb{G}_2\otimes\mathbb{G}_1)\,«\tilde{P}_2,\tilde{P}_1»$.
Proof: By definition, $(\mathbb{G}_1\otimes\mathbb{G}_2)«\tilde{P}_1,\tilde{P}_2»\equiv\mathbb{G}_1'«\tilde{P}_1»\otimes\mathbb{G}_2'«\tilde{P}_2»$ and $(\mathbb{G}_2\otimes\mathbb{G}_1)\,«\tilde{P}_2,\tilde{P}_1»\equiv\mathbb{G}_2«\tilde{P}_2»\otimes\mathbb{G}_1«\tilde{P}_1»$, but by str-SUM2, $\mathbb{G}_1'«\tilde{P}_1»\otimes\mathbb{G}_2'«\tilde{P}_2»\equiv\mathbb{G}_2«\tilde{P}_2»\otimes\mathbb{G}_1«\tilde{P}_1»$, therefore $(\mathbb{G}_1\otimes\mathbb{G}_2)«\tilde{P}_1,\tilde{P}_2»\equiv(\mathbb{G}_2\otimes\mathbb{G}_1)\,«\tilde{P}_2,\tilde{P}_1»$. ■

Corollary 7−97: $\mathbb{G}\otimes\mathbf{0}\equiv\mathbf{0}\otimes\mathbb{G}\equiv\mathbb{G}$.
Proof: By definition, $\mathbb{G}\otimes\mathbf{0}\equiv«\tilde{\eta}»\,(\,\mathbb{G}«\tilde{\eta}»\otimes\mathbf{0}\,)\equiv«\tilde{\eta}»\,(\,\mathbb{G}«\tilde{\eta}»\,)$ and $\mathbf{0}\otimes\mathbb{G}\equiv«\tilde{\eta}»\,(\,\mathbf{0}\otimes\mathbb{G}«\tilde{\eta}»\,)\equiv«\tilde{\eta}»\,(\,\mathbb{G}«\tilde{\eta}»\,)$. ■

Lemma 7−98: $(\mathbb{G}_1\otimes\mathbb{G}_2)\otimes\mathbb{G}_3\equiv\mathbb{G}_1\otimes(\mathbb{G}_2\otimes\mathbb{G}_3)$.
Proof: By definition,
$$\begin{aligned}(\mathbb{G}_1\otimes\mathbb{G}_2)\otimes\mathbb{G}_3 &\equiv«\tilde{\eta}_1,\tilde{\eta}_2,\tilde{\eta}_3»\,(\;«\tilde{\eta}_1,\tilde{\eta}_2»\,(\mathbb{G}_1«\tilde{\eta}_1»\otimes\mathbb{G}_2«\tilde{\eta}_2»\,)\,«\tilde{\eta}_1,\tilde{\eta}_2»\otimes\mathbb{G}_3«\tilde{\eta}_3»\,)\,«\tilde{\eta}_1,\tilde{\eta}_2,\tilde{\eta}_3»\\ &\equiv«\tilde{\eta}_1,\tilde{\eta}_2,\tilde{\eta}_3»\,(\mathbb{G}_1«\tilde{\eta}_1»\otimes\mathbb{G}_2«\tilde{\eta}_2»\otimes\mathbb{G}_3«\tilde{\eta}_3»\,)\,«\tilde{\eta}_1,\tilde{\eta}_2,\tilde{\eta}_3»\\ &\equiv«\tilde{\eta}_1,\tilde{\eta}_2,\tilde{\eta}_3»\,(\mathbb{G}_1«\tilde{\eta}_1»\otimes«\tilde{\eta}_2,\tilde{\eta}_3»(\mathbb{G}_2«\tilde{\eta}_2»\otimes\mathbb{G}_3«\tilde{\eta}_3»\,)\,«\tilde{\eta}_2,\tilde{\eta}_3»\,)\,«\tilde{\eta}_1,\tilde{\eta}_2,\tilde{\eta}_3»\\ &\equiv\mathbb{G}_1\otimes(\mathbb{G}_2\otimes\mathbb{G}_3)\end{aligned}$$
■

Corollary 7−99: Structural equivalence on $\mathbb{G}$ terms is preserved by choice composition, ie., it is a congruence.
Proof: Let $\mathbb{G}_1\equiv\mathbb{G}_1'$ and $\mathbb{G}_2\equiv\mathbb{G}_2'$, then $\mathbb{G}_1\otimes\mathbb{G}_2\equiv«\tilde{\eta}_1,\tilde{\eta}_2»\,(\mathbb{G}_1«\tilde{\eta}_1»\otimes\mathbb{G}_2«\tilde{\eta}_2»\,)\equiv«\tilde{\eta}_1,\tilde{\eta}_2»\,(\mathbb{G}_1'«\tilde{\eta}_1»\otimes\mathbb{G}_2'«\tilde{\eta}_2»\,)\equiv\mathbb{G}_1'\otimes\mathbb{G}_2'$. ■

Proposition 7−100: $\forall G.\exists(\mathbb{G},\tilde{P}).(G\equiv\mathbb{G}«\tilde{P}»)$.
Proof: By induction over the definition of GEC and $\mathbb{G}$ terms:
If $G=\mathbb{G}«\tilde{P}»$, it is self proven;
if $G=\mathbf{0}_{\mathbf{G}}$, let $\mathbb{G}\overset{\text{def}}{=}«\eta»\mathbf{0}_{\mathbf{G}}$ and then we can write $G\equiv\mathbb{G}«\mathbf{0}_{\mathbf{P}}»$;
If $G=B$, select a variable η such that $\eta\notin name(B)$, let $Б\overset{\text{def}}{=}B\{\eta/_{body}(B)\}$ and $\mathbb{B}\overset{\text{def}}{=}«\eta»Б$, then $G\equiv\mathbb{B}«body(B)»$;
If $G=(\nu\,\tilde{n})G_1$, assume G_1 can be represented as $G_1\overset{\text{def}}{=}\mathbb{G}_1«\tilde{P}»$, let $\mathbb{G}\overset{\text{def}}{=}«\tilde{\eta}»\,(\nu\,\tilde{n})\mathbb{G}_1«\tilde{\eta}»$, then $G\equiv\mathbb{G}«\tilde{P}»$;
If $G=G_1\otimes G_2$, assume $G_1\overset{\text{def}}{=}\mathbb{G}_1«\tilde{P}_1»$ and $G_2\overset{\text{def}}{=}\mathbb{G}_2«\tilde{P}_2»$, let $\mathbb{G}\overset{\text{def}}{=}(\mathbb{G}_1\otimes\mathbb{G}_2\,)$ then $G\equiv\mathbb{G}«\tilde{P}_1,\tilde{P}_2»$. ■

The equivenlance relation between $\mathbb{G}$ terms can also be defined in term of transition:

Definition 7-101 (**input commitment of** $\mathcal{G}$): Let $\mathcal{G}\overset{\text{def}}{=}«\tilde{\eta}»\mathcal{G}_1«\tilde{\eta}»$, we say that $\mathcal{G}$ can commit on polar ${}^{\dagger}m$, denote as $\mathcal{G}\downarrow{}^{\dagger}m$, or say that $\mathcal{G}$ can commit on commubication action ${}^{\dagger}m(\tilde{u})$, denote as $\mathcal{G}\downarrow{}^{\dagger}m(\tilde{u})$, if $«\tilde{\eta}»\lfloor\,\rceil\circ[\mathcal{G}«\tilde{\eta}»]\xrightarrow{{}^{\dagger}m(\tilde{u})}«\tilde{\eta}»$
$(\nu\,\tilde{n})(\Lambda\circ[\mathcal{G}_1«\tilde{\eta}»]\,|\,((\tilde{w})\eta_i)\langle\tilde{n},\tilde{u}\rangle)$, where $\eta_i\in\tilde{\eta}$, $\mathit{arity}(\tilde{w})=\mathit{arity}(\tilde{n},\tilde{u})$.

Definition 7-102 ($\mathcal{G}$-**bisimulation**): A symmetric relation $\mathcal{S}$ on $\mathcal{G}$ terms is a $\mathcal{G}$-bisimulation if whenever $\mathcal{G}_1\mathcal{S}\mathcal{G}_2$ then $\mathit{arity}(\mathcal{G}_1)\equiv\mathit{arity}(\mathcal{G}_2)$; and
$«\tilde{\eta}»\lfloor\,\rceil\circ[\mathcal{G}_1«\tilde{\eta}»]\xrightarrow{{}^{\dagger}m(\tilde{u})}«\tilde{\eta}»(\nu\,\tilde{n})(\Lambda\circ[\mathcal{G}_1«\tilde{\eta}»]\,|\,((\tilde{w})\eta_i)\langle\tilde{n},\tilde{u}\rangle)$, where $\eta_i\in\tilde{\eta}$ and $\mathit{arity}(\tilde{w})=\mathit{arity}(\tilde{n},\tilde{u})$,
implies $«\tilde{\eta}»\lfloor\,\rceil\circ[\mathcal{G}_2«\tilde{\eta}»]\xrightarrow{{}^{\dagger}m(\tilde{u})}«\tilde{\eta}»(\nu\,\tilde{n})(\Lambda\circ[\mathcal{G}_2«\tilde{\eta}»]\,|\,((\tilde{w})\eta_i)\langle\tilde{n},\tilde{u}\rangle)$.

Let $\sim_{\mathcal{G}}$ be the largest $\mathcal{G}$-bisimulation. There is no weak version bisimulation for $\mathcal{G}$ terms.

Notation 7-103: For any binary relation $\mathcal{R}$ of process terms, we use $\{P_{i\in I}\}\mathcal{R}\{Q_{i\in I}\}$, or its abbreviation $\tilde{P}\mathcal{R}\tilde{Q}$, to denote $\forall_{i\in I}.(P_i\mathcal{R}Q_i)$.

Lemma 7-104: $\mathcal{G}$-bisimulation is an equivalence.

Proof: Reflexive: $\mathcal{G}\sim_{\mathcal{G}}\mathcal{G}$ for any $\mathcal{G}$, from the definition of $\mathcal{G}$-bisimulation.

Symmetric: if $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$ then $\mathcal{G}_2\sim_{\mathcal{G}}\mathcal{G}_1$, by the definition of $\mathcal{G}$-bisimulation.

Transitive: that is, whenever $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$ and $\mathcal{G}_2\sim_{\mathcal{G}}\mathcal{G}_3$, then $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_3$. To prove this, let $\mathcal{G}_1\mathcal{S}_1\mathcal{G}_2$ and $\mathcal{G}_2\mathcal{S}_2\mathcal{G}_3$, where $\mathcal{S}_1\subseteq\sim_{\mathcal{G}}$ and $\mathcal{S}_2\subseteq\sim_{\mathcal{G}}$, and therefore $\mathcal{G}_1(\mathcal{S}_1\mathcal{S}_2)\mathcal{G}_3$.

Let $«\tilde{\eta}»\lfloor\,\rceil\circ[\mathcal{G}_1«\tilde{\eta}»]\xrightarrow{{}^{\dagger}m(\tilde{u})}«\tilde{\eta}»(\Lambda\circ[\mathcal{G}_1«\tilde{\eta}»]\,|\,\eta_i\langle\tilde{u}\rangle)$, where $\eta_i\in\{\tilde{\eta}\}$, by $\mathcal{G}_1\mathcal{S}_1\mathcal{G}_2$, it implies
$«\tilde{\eta}»\lfloor\,\rceil\circ[\mathcal{G}_2«\tilde{\eta}»]\xrightarrow{{}^{\dagger}m(\tilde{u})}«\tilde{\eta}»(\Lambda\circ[\mathcal{G}_2«\tilde{\eta}»]\,|\,\eta_i\langle\tilde{u}\rangle)$. By $\mathcal{G}_2\mathcal{S}_2\mathcal{G}_3$, it further implies
$«\tilde{\eta}»\lfloor\,\rceil\circ[\mathcal{G}_3«\tilde{\eta}»]\xrightarrow{{}^{\dagger}m(\tilde{u})}«\tilde{\eta}»(\Lambda\circ[\mathcal{G}_3«\tilde{\eta}»]\,|\,\eta_i\langle\tilde{u}\rangle)$, and $\mathcal{G}_2\mathcal{S}_2\mathcal{G}_3$.

Therefore, by the definition of $\mathcal{G}$-bisimulation, $(\mathcal{S}_1\mathcal{S}_2)\subseteq\sim_{\mathcal{G}}$. ■

Corresponding to Lemma-6-82, we have the following two corollaries

Lemma 7-105: If $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$ then $\mathit{guard}(\mathcal{G}_1)\equiv\mathit{guard}(\mathcal{G}_2)$ and $\mathit{excl}(\mathcal{G}_1)\cap\mathit{guard}(\mathcal{G}_1)\equiv\mathit{excl}(\mathcal{G}_2)\cap\mathit{guard}(\mathcal{G}_2)$.

Proof: Assume the contraries, then $\mathcal{G}_1\nsim_{\mathcal{G}}\mathcal{G}_2$ for any of $\mathit{guard}(\mathcal{G}_1)\neq\mathit{guard}(\mathcal{G}_2)$ or $\mathit{excl}(\mathcal{G}_1)\cap\mathit{guard}(\mathcal{G}_1)\neq\mathit{excl}(\mathcal{G}_2)\cap\mathit{guard}(\mathcal{G}_2)$ or $\mathit{lockset}(\mathcal{G}_1)\neq\mathit{lockset}(\mathcal{G}_2)$. ■

Lemma 7-106: If $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$ then $(\mathcal{G}_1«\tilde{P}»)\sim_{gG}(\mathcal{G}_2«\tilde{P}»)$ for any $\tilde{P}$ satisfying $\mathit{arity}(\tilde{P})\equiv\mathit{arity}(\mathcal{G}_1)$.
Proof: Directly from definitions. ■

Proposition 7-107: If $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$ and $\tilde{P}_1\mathcal{S}_{\mathcal{P}}\tilde{P}_2$, then $\Lambda\circ[\mathcal{G}_1«\tilde{P}_1»]\,\mathcal{S}_{\mathcal{P}}\,\Lambda\circ[\mathcal{G}_2«\tilde{P}_2»]$ for all Λ and $\mathcal{S}_{\mathcal{P}}$.

Lemma 7-108: If $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$ then $\mathcal{G}_1\otimes\mathcal{G}\sim_{\mathcal{G}}\mathcal{G}_2\otimes\mathcal{G}$ and $\mathcal{G}\otimes\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}\otimes\mathcal{G}_2$ for any $\mathcal{G}$.
Proof: Lets only prove that for $\mathcal{G}_1\otimes\mathcal{G}\sim_{\mathcal{G}}\mathcal{G}_2\otimes\mathcal{G}$, since $\mathcal{G}\otimes\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}\otimes\mathcal{G}_2$ can be proven in the same way.

From definitions we have $\mathit{arity}(\mathcal{G}_1)\equiv\mathit{arity}(\mathcal{G}_2)$, $\mathcal{G}_1\otimes\mathcal{G}\equiv«\tilde{\eta}_1,\tilde{\eta}»(\mathcal{G}_1«\tilde{\eta}_1»\otimes\mathcal{G}«\tilde{\eta}»)$ and $\mathcal{G}_2\otimes\mathcal{G}\overset{\text{def}}{=}«\tilde{\eta}_1,\tilde{\eta}»(\mathcal{G}_2«\tilde{\eta}_1»\otimes\mathcal{G}«\tilde{\eta}»)$. Assume $(\mathcal{G}_1\otimes\mathcal{G})\downarrow{}^{\dagger}m(\tilde{u})$, then $«\tilde{\eta}_1,\tilde{\eta}»\lfloor\,\rceil\circ[\mathcal{G}_1«\tilde{\eta}_1»\otimes\mathcal{G}«\tilde{\eta}»]\xrightarrow{{}^{\dagger}m(\tilde{u})}«\tilde{\eta}_1,\tilde{\eta}»(\Lambda\circ[\mathcal{G}_1«\tilde{\eta}_1»\otimes\mathcal{G}«\tilde{\eta}»]\,|\,\eta_i\langle\tilde{u}\rangle)$. But this always implies $«\tilde{\eta}_1,\tilde{\eta}»\lfloor\,\rceil\circ[\mathcal{G}_2«\tilde{\eta}_1»\otimes\mathcal{G}«\tilde{\eta}»]\xrightarrow{{}^{\dagger}m(\tilde{u})}«\tilde{\eta}_1,\tilde{\eta}»(\Lambda\circ[\mathcal{G}_2«\tilde{\eta}_1»\otimes\mathcal{G}«\tilde{\eta}»]\,|\,\eta_i\langle\tilde{u}\rangle)$: If $\mathcal{G}_1\downarrow{}^{\dagger}m(\tilde{u})$, then we got it from $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$, and we have $\eta_i\in\{\tilde{\eta}_1\}$; If $\mathcal{G}\downarrow{}^{\dagger}m(\tilde{u})$, then it is self satisfied with $\eta_i\in\{\tilde{\eta}\}$. ■

Lemma 7-109: If $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$ and $\mathcal{G}_3\sim_{\mathcal{G}}\mathcal{G}_4$ then $\mathcal{G}_1\otimes\mathcal{G}_3\sim_{\mathcal{G}}\mathcal{G}_2\otimes\mathcal{G}_4$.

Proof: Apply Lemma-7-108 to $\mathcal{G}_1\sim_{\mathcal{G}}\mathcal{G}_2$ and $\mathcal{G}_3\sim_{\mathcal{G}}\mathcal{G}_4$ then we got $\mathcal{G}_1\otimes\mathcal{G}_3\sim_{\mathcal{G}}\mathcal{G}_2\otimes\mathcal{G}_3$ and $\mathcal{G}_2\otimes\mathcal{G}_3\sim_{\mathcal{G}}\mathcal{G}_2\otimes\mathcal{G}_4$ respectivily, since $\sim_{\mathcal{G}}$ is transitive by Lemma-7-104, we have $\mathcal{G}_1\otimes\mathcal{G}_3\sim_{\mathcal{G}}\mathcal{G}_2\otimes\mathcal{G}_4$. ■

Lemma 7−110: $\mathbb{G}$-bisimulation is a congruence.

Corollary 7−111: $\langle\!\langle\eta\rangle\!\rangle(\text{Б}\otimes\text{Б})\sim_{\mathbb{G}}\langle\!\langle\eta\rangle\!\rangle\text{Б}$, where process variable η is freely appeared in Б.

As an example, it is easy to see

$$\langle\!\langle\eta_a,\eta_b\rangle\!\rangle(\,!(\nu\,\kappa){}^{\dagger}m_a(\tilde{x})L_a.\eta_a\langle\tilde{x},\kappa\rangle\otimes!(\nu\,\kappa){}^{\dagger}m_a(\tilde{x})L_a.\eta_a\langle\tilde{x},\kappa\rangle\otimes!(\nu\,\kappa){}^{\dagger}m_b(\tilde{x})L_b.\eta_b\langle\tilde{x},\kappa\rangle)\sim_{\mathbb{G}}$$
$$\langle\!\langle\eta_a,\eta_b\rangle\!\rangle(\,!(\nu\,\kappa){}^{\dagger}m_a(\tilde{x})L_a.\eta_a\langle\tilde{x},\kappa\rangle\otimes!(\nu\,\kappa){}^{\dagger}m_b(\tilde{x})L_b.\eta_b\langle\tilde{x},\kappa\rangle)$$

7.2 Normalisation operator for higher-ordered GEC choice terms

Now, lets go back to the example where we raised the problem in (7−1). Let

$$\mathbb{G}\overset{\text{def}}{=}\langle\!\langle\eta_{\text{a}},\eta_{\text{b}}\rangle\!\rangle(\,!(\nu\,\kappa){}^{\dagger}m_a(\tilde{x})L_a.\eta_a\langle\tilde{x},\kappa\rangle\otimes!(\nu\,\kappa){}^{\dagger}m_b(\tilde{x})L_b.\eta_b\langle\tilde{x},\kappa\rangle)\quad\text{and}\quad P\overset{\text{def}}{=}\lfloor\,\rfloor\circ[!(\nu\,\kappa){}^{\dagger}n_a(\tilde{x},\kappa).P_a\otimes!(\nu\,\kappa){}^{\dagger}n_b(\tilde{x},\kappa).P_b]$$

Then we can write $G_1\overset{\text{def}}{=}\mathbb{G}\langle\!\langle P_a,P_b\rangle\!\rangle$ and $G_2\overset{\text{def}}{=}\mathbb{G}\langle\!\langle(\tilde{x},\kappa)\bar{n}_a\langle\tilde{x},\kappa\rangle,(\tilde{x},\kappa)\bar{n}_b\langle\tilde{x},\kappa\rangle\rangle\!\rangle$, therefore $\Lambda\circ[G_1]\,|\,P\approx_{\text{g}}\Lambda\circ[G_2]$, that is,

$$\Lambda\circ[\mathbb{G}\langle\!\langle(\tilde{x},\kappa)\bar{n}_a\langle\tilde{x},\kappa\rangle,(\tilde{x},\kappa)\bar{n}_b\langle\tilde{x},\kappa\rangle\rangle\!\rangle]\,|\,\lfloor\,\rfloor\circ[\,!{}^{\dagger}n_a(\tilde{x},\kappa).P_a\otimes!{}^{\dagger}n_b(\tilde{x},\kappa).P_b]\approx_{\text{g}}\Lambda\circ[\mathbb{G}\langle\!\langle P_a,P_b\rangle\!\rangle].$$

If we consider $O_1\overset{\text{def}}{=}\Lambda\circ[G_1]\,|\,P$ and $O_2\overset{\text{def}}{=}\Lambda\circ[G_2]$ are modelling two different versions of objects, then we can naturally think that, the "header" $\mathbb{G}$ defines the exclusive behaviours among the object methods, the "body" $\{P_a,P_b\}$ define the functionality and the "adapter" $\{n_a,n_b\}$ connects them. As we have pointed out before, Λ acts as a thread monitor, so we have isolated some differents aspects at separated componants, that is the great deal.

Similarly, for the example in the equation 7-2, we now can rewrite the processes Q_1 and Q_2 as:

$$Q_1\overset{\text{def}}{=}(m_a,m_b)(\nu\,\kappa_1)\,(\lfloor\check{\kappa}_1@[{}^{\dagger}m_b]\rfloor\circ[\mathbb{G}_1\langle{}^{\dagger}m_a,{}^{\dagger}m_b\rangle\langle\!\langle P_1,P_2\rangle\!\rangle]\,|\,\hat{\kappa}_1)$$
$$Q_2\overset{\text{def}}{=}(m_a,m_b)(\nu\,\kappa_2)\,(\lfloor\check{\kappa}_2@[{}^{\dagger}m_b]\rfloor\circ[\mathbb{G}_2\langle{}^{\dagger}m_a,{}^{\dagger}m_b\rangle\langle\!\langle P_1,P_2\rangle\!\rangle]\,|\,\hat{\kappa}_2),$$

where
$$\mathbb{G}_1\overset{\text{def}}{=}({}^{\dagger}y,{}^{\ddagger}z)\,\langle\!\langle\eta_{\text{a}},\eta_{\text{b}}\rangle\!\rangle(\,!(\nu\,\kappa_a){}^{\dagger}y(\tilde{x})\check{\kappa}_a@J_a.\eta_a\otimes!(\nu\,\kappa_b){}^{\ddagger}z(\tilde{x})\check{\kappa}_b@J_b.\eta_b\otimes!(\nu\,\kappa_b){}^{\ddagger}z(\tilde{x})\check{\kappa}_b@J_b.\eta_b)$$
$$\mathbb{G}_2\overset{\text{def}}{=}({}^{\dagger}y,{}^{\ddagger}z)\,\langle\!\langle\eta_{\text{a}},\eta_{\text{b}}\rangle\!\rangle(\,!(\nu\,\kappa_a){}^{\dagger}y(\tilde{x})\check{\kappa}_a@J_a.\eta_a\otimes!(\nu\,\kappa_b){}^{\ddagger}z(\tilde{x})\check{\kappa}_b@J_b.\eta_b)$$

and we have $\mathbb{G}_1\sim_{\mathbb{G}}\mathbb{G}_2$ and $Q_1\approx_{\text{g}}Q_2$. We can also consider that the processes Q_1 and Q_2 are modelling two versions of the same object in a status where one of its two methods, m_b, is locked forever and another method, m_a, is still available.

In general, with the higher order terms, we may model an object in form of $\Lambda\circ[\mathbb{G}\langle\!\langle\tilde{P}\rangle\!\rangle]$ or a composition of such., which will be discussed in details in section 11.

However, there still a problem left: the κ-calculus does not have the composition power on exclusion yet as Noble's algebra of exclusion does. For example, in the latter the composition of separately defined relations $m_1\times m_2$, $m_2\times m_3$, $m_3\times m_1$, and $m_2\times m_2$ will give the componded relation $m_1\times\overline{m_2}\times m_3$. But this cannot be achieved directly in our κ-calculus by composing $\mathbb{G}_1\otimes\mathbb{G}_2\otimes\mathbb{G}_3\otimes\mathbb{G}_4$ for

$$\mathbb{G}_1\overset{\text{def}}{=}\langle\!\langle\tilde{\eta}\rangle\!\rangle(\,!(\nu\,\kappa){}^{\dagger}m_1\,\check{\kappa}@[{}^{\dagger}m_2].\eta_1\otimes!(\nu\,\kappa){}^{\dagger}m_2\,\check{\kappa}@[{}^{\dagger}m_1].\eta_2\,)$$
$$\mathbb{G}_2\overset{\text{def}}{=}\langle\!\langle\tilde{\eta}\rangle\!\rangle(\,!(\nu\,\kappa){}^{\dagger}m_2\,\check{\kappa}@[{}^{\dagger}m_3].\eta_2\otimes!(\nu\,\kappa){}^{\dagger}m_3\,\check{\kappa}@[{}^{\dagger}m_2].\eta_3\,)$$
$$\mathbb{G}_3\overset{\text{def}}{=}\langle\!\langle\tilde{\eta}\rangle\!\rangle(\,!(\nu\,\kappa){}^{\dagger}m_3\,\check{\kappa}@[{}^{\dagger}m_1].\eta_3\otimes!(\nu\,\kappa){}^{\dagger}m_1\,\check{\kappa}@[{}^{\dagger}m_3].\eta_1\,)$$
$$\mathbb{G}_4\overset{\text{def}}{=}\langle\!\langle\tilde{\eta}\rangle\!\rangle\;!(\nu\,\kappa){}^{\dagger}m_2\,\check{\kappa}@[{}^{\dagger}m_2].\eta_2.$$

Furthermore, what the behaviour we would expect from an object if expressions such as

$${}^{\dagger}m(\tilde{x})L_1.\eta\otimes{}^{\dagger}m(\tilde{x})L_2.\eta\quad\text{or}\quad{}^{\dagger}m(\tilde{x})L.\eta_1\otimes{}^{\dagger}m(\tilde{x})L.\eta_2$$

is included? To enable the calculus expressing objects model more compositively and clear, some restrictions on it are necessary. However, to avoid much more complicated reduction rules and bisimulation relateds, we prefer do not modify the basic syntax of the calculus, instead, we modify the way of modelling objects.

Definition 7-112: An higher order GEC choice $\mathbb{G}$ is *canonical*, denoted as $\overline{\mathbb{G}}$, if it satisfies that, either $\mathbb{G}=\mathbf{0}$ or it can be presented in the form $\mathbb{G}\overset{\text{def}}{=}\mathbb{B}\otimes\mathbb{G}_1$ where:

1. $\mathbb{G}_1$ is a canonical higher order GEC choice (and therefore we may also write it as $\overline{\mathbb{G}}_1$); and
2. $guard(\mathbb{B})\notin guard(\mathbb{G}_1)$; and
3. The lock key of $\mathbb{B}$ is defined local to $\mathbb{B}$, that is, it is in the form of $\mathbb{B}\overset{\text{def}}{=}«\eta»\,!(\nu\,\kappa)^{\dagger}m(\tilde{x})\,\check{\kappa}@J\,.\,\eta$.

If $\mathbb{B}$ is the only branch of $\overline{\mathbb{G}}$, we may also use symbol $\overline{\mathbb{B}}$ to denote $\overline{\mathbb{G}}$.

Corollary 7-113: For all $\mathbb{B}_i,\mathbb{B}_j\in branch(\overline{\mathbb{G}})$, i≠j implies $guard(\mathbb{B}_i)\neq guard(\mathbb{B}_j)$. That is, a canonical higher order GEC choice has the form: $\overline{\mathbb{G}}\overset{\text{def}}{=}«\tilde{\eta}»\bigotimes_{i\in I}\,!(\nu\,\kappa)^{\dagger}m_i(\tilde{x})\,\check{\kappa}@J_i\,.\,\eta_i$ where $\forall_{i,j\in I}.($ i≠j implies $^{\dagger}m_i\neq{}^{\dagger}m_j)$.

For example, $\mathbb{G}\overset{\text{def}}{=}«\eta_1,\eta_2»\,!(\nu\,\kappa)^{\dagger}m_1(\tilde{x})\,\check{\kappa}@J_1.\,\eta_1\otimes\,!(\nu\,\kappa)^{\dagger}m_2(\tilde{x})\,\check{\kappa}@J_2.\,\eta_2$ is canonical.

Lemma 7-114: If $(\mathbb{G}_1\otimes\mathbb{G}_2)$ is a canonical higher order GEC choice, then both $\mathbb{G}_1,\mathbb{G}_2$ are also canonical higher order GEC choices, and $guard(\mathbb{G}_1)\cap guard(\mathbb{G}_2)=\varnothing$.

Proof: By the definition, if otherwise then $(\mathbb{G}_1\otimes\mathbb{G}_2)$ cannot be a canonical higher order GEC choice. ■

To simplify descriptions when studying the properties of canonical higher order GEC choices, we introduce some more auxiliary functions.

Definition 7-115: The function $rm(\overline{\mathbb{G}},\tilde{m})$ presents a canonical higher order GEC choice which is obtained by removing from the given canonical higher order GEC choice $\overline{\mathbb{G}}$ all branches of a guard in $\tilde{m}$, without change the order of remaining branches. Morm formally, $rm(\overline{\mathbb{G}},\tilde{m})$ is definied by:

$$rm(\mathbf{0},\tilde{m})\quad\overset{\text{def}}{=}\quad\mathbf{0};$$

$$rm(\overline{\mathbb{B}}\otimes\overline{\mathbb{G}},\tilde{m})\quad\overset{\text{def}}{=}\begin{cases}rm(\overline{\mathbb{G}},\tilde{m}), & \text{if}\quad guard(\overline{\mathbb{B}})\in\tilde{m},\\ \overline{\mathbb{B}}\otimes rm(\overline{\mathbb{G}},\tilde{m}), & \text{if}\quad guard(\overline{\mathbb{B}})\notin\tilde{m}.\end{cases}$$

The function $ints(\overline{\mathbb{G}},\tilde{m})\overset{\text{def}}{=}rm(\overline{\mathbb{G}},guards(\overline{\mathbb{G}})-\tilde{m})$ presents a canonical higher order GEC choice which is obtained by removing from the given canonical higher order GEC choice $\overline{\mathbb{G}}$ all branches of a guard not in $\tilde{m}$.

Corollary 7-116:
1. $rm(\overline{\mathbb{G}},\tilde{m})\equiv ints(\overline{\mathbb{G}},guards(\overline{\mathbb{G}})-\tilde{m})$;
2. $guards(rm(\overline{\mathbb{G}},\tilde{m}))\equiv guards(\overline{\mathbb{G}})-\tilde{m}$;
3. $guards(ints(\overline{\mathbb{G}},\tilde{m}))\equiv guards(\overline{\mathbb{G}})\cap\tilde{m}$;
4. $rm(rm(\overline{\mathbb{G}},\tilde{m}),\tilde{n})\equiv rm(\overline{\mathbb{G}},\tilde{m}\cup\tilde{n})$;
5. $ints(ints(\overline{\mathbb{G}},\tilde{m}),\tilde{n})\equiv ints(\overline{\mathbb{G}},\tilde{m}\cap\tilde{n})$.

Proof: The proofs for 2-4 are trivial, here we only show that for 1 and 5.

1. $rm(\overline{\mathbb{G}},\tilde{m})\equiv rm(\overline{\mathbb{G}},guards(\overline{\mathbb{G}})-(guards(\overline{\mathbb{G}})-\tilde{m}))\equiv ints(\overline{\mathbb{G}},guards(\overline{\mathbb{G}})-\tilde{m})$;

5.
$$\begin{aligned}ints(ints(\overline{\mathbb{G}},\tilde{m}),\tilde{n})&\equiv rm(ints(\overline{\mathbb{G}},\tilde{m}),(guards(\overline{\mathbb{G}})\cap\tilde{m})-\tilde{n})\\&\equiv rm(rm(\overline{\mathbb{G}},guards(\overline{\mathbb{G}})-\tilde{m}),(guards(\overline{\mathbb{G}})\cap\tilde{m})-\tilde{n})\\&\equiv rm(\overline{\mathbb{G}},(guards(\overline{\mathbb{G}})-\tilde{m})\cup(guards(\overline{\mathbb{G}})\cap\tilde{m})-\tilde{n})\\&\equiv rm(\overline{\mathbb{G}},guards(\overline{\mathbb{G}})-(guards(\overline{\mathbb{G}})\cap\tilde{m}\cap\tilde{n}))\\&\equiv rm(\overline{\mathbb{G}},guards(\overline{\mathbb{G}})-(\tilde{m}\cap\tilde{n}))\\&\equiv ints(\overline{\mathbb{G}},\tilde{m}\cap\tilde{n}).\end{aligned}$$
■

Definition 7-117: The combination operator ⊞, mapping a pair of canonical higher order GEC choices to a single canonical higher order GEC choice, is defined as:

1). $\mathbf{0}\boxplus\overline{\mathbb{G}}\quad\overset{\text{def}}{=}\quad\overline{\mathbb{G}}$;

2). $\overline{\mathbb{G}}\boxplus\mathbf{0}\quad\overset{\text{def}}{=}\quad\overline{\mathbb{G}}$;

3). $$\overline{\mathbb{B}}_1\boxplus\overline{\mathbb{B}}_2\quad\overset{\text{def}}{=}\begin{cases}\overline{\mathbb{B}}_1\otimes\overline{\mathbb{B}}_2, & \text{if}\quad guard(\overline{\mathbb{B}}_1)\neq guard(\overline{\mathbb{B}}_2),\\ «\eta»\,(!(\nu\,\kappa)^{\dagger}m(\tilde{x})\,\check{\kappa}@(excl(\overline{\mathbb{B}}_1)\cup excl(\overline{\mathbb{B}}_2)).\eta), & \text{if}\quad m=guard(\overline{\mathbb{B}}_1)=guard(\overline{\mathbb{B}}_2);\end{cases}$$

4). $(\overline{\mathbb{B}}\otimes\overline{\mathbb{G}}_1)\boxplus\overline{\mathbb{G}}_2\quad\overset{\text{def}}{=}\quad\overline{\mathbb{B}}\otimes(\overline{\mathbb{G}}_1\boxplus\overline{\mathbb{G}}_2)$ if $guard(\overline{\mathbb{B}})\notin guard(\overline{\mathbb{G}}_2)$;

5). $(\overline{\mathbb{B}}_1\otimes\overline{\mathbb{G}}_1)\boxplus(\overline{\mathbb{G}}'_2\otimes\overline{\mathbb{B}}_2\otimes\overline{\mathbb{G}}''_2)\overset{\text{def}}{=}(\overline{\mathbb{B}}_1\boxplus\overline{\mathbb{B}}_2)\otimes(\overline{\mathbb{G}}_1\boxplus(\overline{\mathbb{G}}'_2\otimes\overline{\mathbb{G}}''_2))$ if $guard(\overline{\mathbb{B}}_1)=guard(\overline{\mathbb{B}}_2)$.

Note, because the operands at both sides of the $\boxplus$ operator should be canonical higher order GEC choices, in the clause 5) of the definition, we have $guard(\overline{\mathcal{B}}_1) \notin (guard(\overline{\mathcal{G}}_1) \cup guard(\overline{\mathcal{G}}'_2) \cup guard(\overline{\mathcal{G}}''_2))$. Therefore the clause 1) to 5) in the above definition have covered all possible combinations of the operands.

Corollary 7−118: $\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_2 \equiv (\overline{\mathcal{G}}_1 \boxplus ints(\overline{\mathcal{G}}_2, guards(\overline{\mathcal{G}}_1))) \otimes rm(\overline{\mathcal{G}}_2, guards(\overline{\mathcal{G}}_1))$;
$rm(\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_2, \tilde{m}) \equiv rm(\overline{\mathcal{G}}_1, \tilde{m}) \boxplus rm(\overline{\mathcal{G}}_2, \tilde{m})$, $ints(\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_2, \tilde{m}) \equiv ints(\overline{\mathcal{G}}_1, \tilde{m}) \boxplus ints(\overline{\mathcal{G}}_2, \tilde{m})$
Proof: Both the proofs are trivial, obtained by simple induction over the definition of the $\boxplus$ operator. ■

Lemma 7−119: Given any canonical higher order GEC choice $\overline{\mathcal{G}}_1$ and $\overline{\mathcal{G}}_2$, then $(\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_2)$ is always canonical.
Proof: This is also a trivial proof obtained by induction over the definition of the $\boxplus$ operator, since the 5) clause eliminates any re-appearing of a guard. ■

Lemma 7−120: For canonical higher order GEC choices $\overline{\mathcal{G}}_1$, $\overline{\mathcal{G}}_2$ and $\overline{\mathcal{G}}$, if $\overline{\mathcal{G}}_1 \sim_{\mathcal{G}} \overline{\mathcal{G}}_2$ then $(\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}) \sim_{\mathcal{G}} (\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}})$ and $(\overline{\mathcal{G}} \boxplus \overline{\mathcal{G}}_1) \sim_{\mathcal{G}} (\overline{\mathcal{G}} \boxplus \overline{\mathcal{G}}_2)$.

Proof: Induction over definition of $\boxplus$ by applying Lemma-7-108. ■

Lemma 7−121: For canonical higher order GEC choices $\overline{\mathcal{G}}_1, \overline{\mathcal{G}}_2, \overline{\mathcal{G}}_3$ and $\overline{\mathcal{G}}_2$, if $\overline{\mathcal{G}}_1 \sim_{\mathcal{G}} \overline{\mathcal{G}}_2$ and $\overline{\mathcal{G}}_3 \sim_{\mathcal{G}} \overline{\mathcal{G}}_4$ then $(\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_3) \sim_{\mathcal{G}} (\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_4)$.

Proof: By the previous corollary, $(\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_3) \sim_{\mathcal{G}} (\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_3)$ and $(\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_3) \sim_{\mathcal{G}} (\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_4)$, therefore $(\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_3) \sim_{\mathcal{G}} (\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_4)$. ■

Lemma 7−122: $\overline{\mathcal{G}}_1 \boxplus (\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_3) \equiv (\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_2) \boxplus \overline{\mathcal{G}}_3$ $\quad (\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_2) \boxplus \overline{\mathcal{G}}_3 \equiv \overline{\mathcal{G}}_1 \boxplus (\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_3)$ for all canonical higher order GEC choice $\overline{\mathcal{G}}_1$, $\overline{\mathcal{G}}_2$ and $\overline{\mathcal{G}}_3$.
Proof: By the definition of $\boxplus$ and by Corollary-7-116, we have

$$\begin{aligned}(\overline{\mathcal{G}}_1 \boxplus \overline{\mathcal{G}}_2) \boxplus \overline{\mathcal{G}}_3 \equiv\ & (\overline{\mathcal{G}}_1 \boxplus ints(\overline{\mathcal{G}}_2, guards(\overline{\mathcal{G}}_1)) \boxplus ints(\overline{\mathcal{G}}_3, guards(\overline{\mathcal{G}}_1))) \\ & \otimes (rm(\overline{\mathcal{G}}_2, guards(\overline{\mathcal{G}}_1)) \boxplus ints(\overline{\mathcal{G}}_3, guards(\overline{\mathcal{G}}_2) - guards(\overline{\mathcal{G}}_1))) \\ & \otimes rm(\overline{\mathcal{G}}_3, guards(\overline{\mathcal{G}}_2) \cup guards(\overline{\mathcal{G}}_1))\end{aligned}$$

B y Corollary-7-118 and Corollary-7-116,

$$\begin{aligned}\overline{\mathcal{G}}_1 \boxplus (\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_3) &\equiv (\overline{\mathcal{G}}_1 \boxplus ints((\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_3), guards(\overline{\mathcal{G}}_1))) \otimes rm((\overline{\mathcal{G}}_2 \boxplus \overline{\mathcal{G}}_3), guards(\overline{\mathcal{G}}_1)) \\ &\equiv (\overline{\mathcal{G}}_1 \boxplus ints(\overline{\mathcal{G}}_2, guards(\overline{\mathcal{G}}_1)) \boxplus ints(\overline{\mathcal{G}}_3, guards(\overline{\mathcal{G}}_1))) \\ &\quad \otimes (rm(\overline{\mathcal{G}}_2, guards(\overline{\mathcal{G}}_1)) \boxplus ints(\overline{\mathcal{G}}_3, guards(\overline{\mathcal{G}}_2) - guards(\overline{\mathcal{G}}_1))) \\ &\quad \otimes rm(\overline{\mathcal{G}}_3, guards(\overline{\mathcal{G}}_2) \cup guards(\overline{\mathcal{G}}_1))\end{aligned}$$ ■

From now on we restricte that no non-canonical GEC choice should be used in an object model, therefore the statement about object model earlier in this subsection should re-addressed as: In general, an object is modelled in form of $\Lambda^{\circ}[\overline{\mathcal{G}}\langle\!\langle \tilde{P} \rangle\!\rangle]$ or a composition of such. More detailed discussion about object modelling will be in section 11.

8 A simple type system

To displine the modelling of proecess, a simple type system, close to that in [Liu97], is included in the κ-calculus. A term H having type T is denoted as H:T. The *first-order types*, ranged over by ι, given by

$$\iota ::= \lambda \mid \delta \mid \beta, \qquad \lambda ::= \overset{+}{\lambda} \mid \bar{\lambda}, \qquad \delta ::= \overset{+}{\delta} \mid \bar{\delta},$$

where λ is a set of atomic types called *link sorts*, whose values are communication polars, and can be further devided to two subsets of types, the input link $\overset{+}{\lambda}$ and output link $\bar{\lambda}$; δ is a set of atomic types called *signal sorts*, which has only two atomic types, the input signal type $\overset{+}{\delta}$ and output signal type $\bar{\delta}$, whose values are input and output key polars respectively; β are some basic type such as integers, boolean, etc. However, with the same technique demonstrated by

[Milner96] where basic types values are modelled by names in the π-calculus, we can always model basic types by polars in the κ-calculus. Therefore, in this paper we need not consider any basic types other than polars.

The *higher-order process types*, ranged over by ξ, are given by ξ::=pabs$(\iota_1, \iota_2,\ldots, \iota_n) \mid$ pabs$(\xi_1, \xi_2,\ldots, \xi_n)$;
and the *higher-order choice types*, ranged over by θ, are given by θ::=gabs$(\iota_1, \iota_2,\ldots, \iota_n) \mid$ gabs$(\xi_1, \xi_2,\ldots, \xi_n)$.

To define the idea of *well-typing*, we introduce the sorting function lnk from link sorts to tuples of output link sorts, so that $lnk(\lambda)=(\tilde{\bar{\lambda}})$ permits polars of sort λ to communicate tuples of values of type $\tilde{\bar{\lambda}}$. For simplifying expressions, we use $\overset{+}{\iota}$ to stand a first order type in either $\overset{+}{\lambda}$ or $\overset{+}{\delta}$, and $\bar{\iota}$ to stand a type in either $\bar{\lambda}$ or $\bar{\delta}$. we also use $m{:}\overset{\pm}{\lambda}$ to stand for a pair of communication polars $\overset{+}{m}{:}\overset{+}{\lambda}$ and $\bar{m}{:}\bar{\lambda}$, where $lnk(\overset{+}{\lambda})=lnk(\bar{\lambda})$; use $\kappa{:}\overset{\pm}{\delta}$ to stand for a pair of key polars $\check{\kappa}{:}\overset{+}{\delta}$ and $\hat{\kappa}{:}\bar{\delta}$; and use $n{:}\overset{\pm}{\iota}$ to stand for either $n{:}\overset{\pm}{\lambda}$ or $n{:}\overset{\pm}{\delta}$.

A communication action α is *well-typed* when it is either

1) $\overset{+}{m}(\tilde{u})$, where $\overset{+}{m}{:}\overset{+}{\lambda}$, $lnk(\lambda)=(\tilde{\bar{\lambda}})$ and $\tilde{u}{:}\tilde{\bar{\lambda}}$; or
2) $\bar{m}\langle\tilde{u}\rangle$, where $\bar{m}{:}\bar{\lambda}$, $lnk(\lambda)=(\tilde{\bar{\lambda}})$ and $\tilde{u}{:}\tilde{\bar{\lambda}}$; or
3) $\check{\kappa}{:}\overset{+}{\delta}$; or
4) $\hat{\kappa}{:}\bar{\delta}$.

A lock L is *well-typed* whenever it is in the form:

5) $\check{\kappa}@J$, where $\check{\kappa}{:}\overset{+}{\delta}$ and either $J=\varnothing$, $J=\boldsymbol{M}$ or $J=[\tilde{\overset{+}{m}}]$ and $\tilde{\overset{+}{m}}{:}\tilde{\overset{+}{\lambda}}$.

A locking status Λ is *well-typed* when it is either empty or every element of Λ is well-typed.

6) $\lfloor\,\rfloor$; or
7) $addl(L,\Lambda)$, where L and Λ are well-typed; or
8) Λ/L, where L and Λ are well-typed.

A defining equation $E \overset{\text{def}}{=} R$ is *well-typed* if R has the same type as E. Suppose that each agent variable and agent constant is assigned with a higher-order type, then each well-typed process expression and abstraction acquires a unique types as follows:

9) $\mathbf{0_P}$: pabs( );
10) $\bar{m}\langle\tilde{u}\rangle$: pabs( ), if $\bar{m}\langle\tilde{u}\rangle$ is a well-typed action
11) $\hat{\kappa}$: pabs( ), if $\hat{\kappa}$ is a well-typed action;
12) $\prod_{i\in I} P_i$: pabs( ) if for each $i\in I$, P_i : pabs( );
13) $\Lambda\circ[G]$: pabs( ) if Λ is well-typed and G : gabs( );
14) $(\nu\ \tilde{n})P$: pabs( ) if P: pabs( ) and $\tilde{n}{:}\tilde{\overset{\pm}{\iota}}$;
15) $P\langle\tilde{a}\rangle$: pabs$(\tilde{\iota})$ if P : pabs$(\tilde{\iota}_a, \tilde{\iota})$ and $\tilde{a}{:}\ \tilde{\iota}_a$; and $P\langle\tilde{a}\rangle$: pabs( ) if P : pabs$(\tilde{\iota}_a)$ and $\tilde{a}{:}\ \tilde{\iota}_a$;
16) $(\tilde{w})P$: pabs$(\tilde{\iota}_w, \tilde{\iota})$ if P : pabs$(\tilde{\iota})$ and $\tilde{w}{:}\ \tilde{\iota}_w$; and $(\tilde{w})P$: pabs$(\tilde{\iota}_n)$ if P : pabs( ) and $\tilde{w}{:}\ \tilde{\iota}_w$;
17) $\mathbb{P}$«$\tilde{P}$» : pabs$(\tilde{\xi})$ if $\mathbb{P}$: pabs$(\tilde{\xi}_p, \tilde{\xi})$ and $\tilde{P}{:}\tilde{\xi}_p$; and $\mathbb{P}$«$\tilde{P}$» : pabs( ) if $\mathbb{P}$: pabs$(\tilde{\xi}_p)$ and $\tilde{P}{:}\tilde{\xi}_p$;
18) «$\tilde{\eta}$»$\mathbb{P}$: pabs$(\tilde{\xi})$ if $\mathbb{P}$: pabs( ), and $\tilde{\eta} : \tilde{\xi}$.

and each well-typed GEC expression and abstraction acquires a unique types as follows:

19) $\mathbf{0_G}$: gabs( );
20) $\bigotimes_{i\in I} G_i$: gabs( ) if for each $i\in I$, G_i : gabs( );
21) $\overset{+}{m}(\tilde{x})L.P$: gabs( ) if P : pabs( ) and action $\overset{+}{m}(\tilde{x})$ and lock L are well-typed;
22) $(\nu\ \tilde{n})G$: gabs( ) if G : gabs( ) and $\tilde{n}{:}\tilde{\overset{\pm}{\iota}}$;
23) $G\langle\tilde{a}\rangle$: gabs$(\tilde{\iota})$ if G : gabs$(\tilde{\iota}_a, \tilde{\iota})$ and $\tilde{a}{:}\ \tilde{\iota}_a$; and $G\langle\tilde{a}\rangle$: gabs( ) if G : gabs$(\tilde{\iota}_a)$ and $\tilde{a}{:}\ \tilde{\iota}_a$;
24) $(\tilde{w})G$: gabs$(\tilde{\iota})$ if G : gabs( ) and $\tilde{w}{:}\ \tilde{\iota}$;
25) $\mathbb{G}$«$\tilde{P}$» : gabs( ) if $\mathbb{G}$: gabs$(\tilde{\xi})$, and $\tilde{P} : \tilde{\xi}$;
26) «$\tilde{\eta}$»$\mathbb{H}$: gabs$(\tilde{\xi})$ if $\mathbb{H}$: gabs( ), and $\tilde{\eta} : \tilde{\xi}$.
27) $\mathbb{G}_1\otimes\mathbb{G}_2$: gabs$(\tilde{\xi}_1,\tilde{\xi}_2)$ if $\mathbb{G}_1$: gabs$(\tilde{\xi}_1)$, and $\mathbb{G}_2$: gabs$(\tilde{\xi}_2)$.

Where the clause 15, 17 and 24 base on the fact that the type of $S \overset{\text{def}}{=} (\tilde{w})(((\tilde{x},\tilde{y})R)\ \langle\tilde{a},\tilde{w}\rangle)$ can be determined by that of the partial instanced R.

The type of a canonical higher order GEC choice $\overline{\mathbb{G}}$ is covered by clause 25 and 26, since $\overline{\mathbb{G}}$ is a special case of $\mathbb{G}$. The type of a term $\overline{\mathbb{G}}_1\boxplus\overline{\mathbb{G}}_2$ depends on the expression of both $\overline{\mathbb{G}}_1$ and $\overline{\mathbb{G}}_2$, and can be individually derived by applying the clause 19 and 25 to 27 above to the definition of operator $\boxplus$:

$\mathbf{0}\boxplus\overline{\mathbb{G}}$: $\mathsf{gabs}(\tilde{\xi})$ and $\overline{\mathbb{G}}\boxplus\mathbf{0}$: $\mathsf{gabs}(\tilde{\xi})$, if $\mathbb{G}:\mathsf{gabs}(\tilde{\xi})$;
$\overline{\mathbb{B}}_1\boxplus\overline{\mathbb{B}}_2$: $\mathsf{gabs}(\xi)$, if $\overline{\mathbb{B}}_1:\mathsf{gabs}(\xi)$ and $guard(\overline{\mathbb{B}}_1)=guard(\overline{\mathbb{B}}_2)$;
$\overline{\mathbb{B}}_1\boxplus\overline{\mathbb{B}}_2$: $\mathsf{gabs}(\xi_1,\xi_2)$, if $\overline{\mathbb{B}}_1:\mathsf{gabs}(\xi_1)$, $\overline{\mathbb{B}}_2:\mathsf{gabs}(\xi_2)$ and $guard(\overline{\mathbb{B}}_1)\neq guard(\overline{\mathbb{B}}_2)$;
$(\overline{\mathbb{B}}\otimes\overline{\mathbb{G}}_1)\boxplus\overline{\mathbb{G}}_2$: $\mathsf{gabs}(\xi,\tilde{\xi})$, if $\overline{\mathbb{B}}:\mathsf{gabs}(\xi)$, $\overline{\mathbb{G}}_1\boxplus\overline{\mathbb{G}}_2$: $\mathsf{gabs}(\xi_2)$ and $guard(\overline{\mathbb{B}})\notin guard(\overline{\mathbb{G}}_2)$;
$(\overline{\mathbb{B}}_1\otimes\overline{\mathbb{G}}_1)\boxplus(\overline{\mathbb{G}}'_2\otimes\overline{\mathbb{B}}_2\otimes\overline{\mathbb{G}}''_2)$: $\mathsf{gabs}(\tilde{\xi}_1,\tilde{\xi}_2)$ if $(\overline{\mathbb{B}}_1\boxplus\overline{\mathbb{B}}_2)$: $\mathsf{gabs}(\tilde{\xi}_1)$, and $\overline{\mathbb{G}}_1\boxplus(\overline{\mathbb{G}}'_2\otimes\overline{\mathbb{G}}''_2)$: $\mathsf{gabs}(\tilde{\xi}_2)$.

9 Encoding the π-calculus in the κ-calculus

[Amadio96]κ-calculusκ-calculusκ-calculusκ-calculusκ-calculusκ-calculusBoth standard synchronous π-calculus and the asynchronous π-calculus of [Amadio96] can be directly encoded in the κ-calculus. However, for simplicity and for concentration on the problem we are interested in, here we only discuss the encoding between the polar π-calculus of [Zhang02A] and the κ-calculus. The mapping between the asynchronous π-calculus of [Amadio96] and the polar π-calculus have been given in [Zhang02A], and the mapping the standard synchronous π-calculus and the asynchronous π-calculus has been well known. Combine the three together, where each of them focuses on a difference issue, the full circle of mapping from the standard synchronous π-calculus to κ-calculus will be formed.

9.1 From the polar π-calculus to the κ-calculus

The polar π-calculus of [Zhang02A] is an asynchronous π-calculus with polars, with the following syntax:

$$P ::= \overline{m}\langle\tilde{u}\rangle \;\big|\; (\nu\,\tilde{n})P \;\big|\; P_1 \mid P_2 \;\big|\; !B \;\big|\; G \qquad G ::= \mathbf{0} \;\big|\; B \;\big|\; (\nu\,\tilde{n})G \;\big|\; G_1 + G_2 \qquad B ::= {}^{\dagger}m(\tilde{x}).P$$

Encoding the P-terms of the polar π-calculus in the κ-calculus is straightforward, lets use subscript κ(G) to indicate a term is encoded into a G-term of the κ-calculus, then:

$[\![\overline{m}\langle\tilde{u}\rangle]\!]_\kappa \overset{\text{def}}{=} \overline{m}\langle\tilde{u}\rangle$;
$[\![P_1 \mid P_2]\!]_\kappa \overset{\text{def}}{=} [\![P_1]\!]_\kappa \mid [\![P_2]\!]_\kappa$;
$[\![(\nu\;\tilde{n})P]\!]_\kappa \overset{\text{def}}{=} (\nu\;\tilde{n})\;[\![P]\!]_\kappa$;
$[\![G]\!]_\kappa \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil\,[\![G]\!]_{\kappa(G)}\rfloor$;
$[\![!{}^{\dagger}m(\tilde{x}).P]\!]_\kappa \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil\,!{}^{\dagger}m(\tilde{x})(\nu)@\varnothing.[\![P]\!]_\kappa\rfloor$;

The only serious thing is to encode the "+" operation, which is simply to always enforce full locking:

$[\![{}^{\dagger}m_1(\tilde{x}).P_1 + {}^{\dagger}m_2(\tilde{x}).P_2]\!]_\kappa \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil\,!{}^{\dagger}m_1(\tilde{x})(\nu)@[{}^{\dagger}m_2].P_1 \otimes\, !{}^{\dagger}m_2(\tilde{x})(\nu)@[{}^{\dagger}m_1].P_2\rfloor$;
$[\![\sum_{i\in I}{}^{\dagger}m_i(\tilde{x}).P_i]\!]_\kappa \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil\bigotimes_{i\in I}\,!\,{}^{\dagger}m_i(\tilde{x})(\nu)@[{}^{\dagger}m_{j\in I}].P_i\rfloor$.

This encoding is based on the belief of that even in the π-calculus all the choices must be guardered. However, in the real life it is not uncommon for many writers only keep this belief in mind implicitly. For example, one may use $P_1 + P_2$ to mean $m_1.Q_1 + m_2.Q_{21} + m_3.Q_{31} + m_4.Q_{41} + m_5.Q_5$ for $P_1 \overset{\text{def}}{=} m_1.Q_1 + m_2.Q_2$ and $P_1 \overset{\text{def}}{=} m_3.Q_{31} + m_4.Q_{41} + m_5.Q_5$. To keep these kind of convenience and flexibility, we can take the advantage of the fact that the affect of exclusion set is never beyong the GEC's scope. With the symbol $\boldsymbol{M}$ the set of all channel names with input polar, the encoding of G terms can be written as

$[\![\mathbf{0}]\!]_{\kappa(G)} \overset{\text{def}}{=} \mathbf{0}$;
$[\![{}^{\dagger}m(\tilde{x}).P]\!]_{\kappa(G)} \overset{\text{def}}{=} !{}^{\dagger}m(\tilde{x})(\nu)@\boldsymbol{M}.[\![P]\!]_\kappa$;

$$[\![(\nu\ \tilde{n})G]\!]_{\kappa(G)} \overset{\text{def}}{=} (\nu\ \tilde{n})[\![G]\!]_{\kappa(G)};$$
$$[\![G_1+G_2]\!]_{\kappa(G)} \overset{\text{def}}{=} [\![G_1]\!]_{\kappa(G)} \otimes [\![G_2]\!]_{\kappa(G)};$$

The τ-action is neither used as a prefix in the polar π-calculus nor κ-calculus, therefore is not appeared in the encoding above. [Zhang02A] has given the τ-prefix as an abbreviation of other actions in the polar π-calculus, we can also encode it directly κ-calculus as:

$$[\![\tau.P+G]\!]_\kappa \quad \overset{\text{def}}{=} (\nu\, m)\,(\bar{m} \,|\, \lfloor\,\rfloor\circ[\, !{}^{\dagger}\bar{m}\ (\nu)@\boldsymbol{M}.[\![P]\!]_\kappa \otimes [\![G]\!]_{\kappa(G)}\,]\,);$$
$$[\![!\tau.P]\!]_\kappa \quad \overset{\text{def}}{=} (\nu\, m)\,(\bar{m} \,|\, \lfloor\,\rfloor\circ[\, !{}^{\dagger}\bar{m}(\tilde{\vec{x}})(\nu)@\varnothing\,.\,(\bar{m} \,|\, [\![P]\!]_\kappa)\,]\,);$$

9.2 From the κ-calculus back to the polar π-calculus

Encoding the P-terms of the κ-calculus into polar π-calculus is also very simple:

$$[\![\bar{m}\langle\tilde{\vec{u}}\rangle]\!]_{p\pi} \quad \overset{\text{def}}{=} \bar{m}\langle\tilde{\vec{u}}\rangle;$$
$$[\![P_1 \,|\, P_2]\!]_{p\pi} \quad \overset{\text{def}}{=} [\![P_1]\!]_{p\pi} \,|\, [\![P_2]\!]_{p\pi};$$
$$[\![(\nu\ \tilde{n})P]\!]_{p\pi} \quad \overset{\text{def}}{=} (\nu\ \tilde{n})\ [\![P]\!]_{p\pi};$$

Converting G-terms from κ-calculus to the polar π-calculus or other variations of π-calculi is however a quite difficult or complicated task, if not impossible at all. The composibility of guards will be lost in a π encoding, that is, you may not be able to encode $[\![G_1]\!]_{p\pi}$ for $[\![G_1 \otimes G_2]\!]_{p\pi}$ without knowledges about some encoding details of $[\![G_2]\!]_{p\pi}$. This is one of the major reason why we do need the κ-calculus.

With the difficulty in encoding the κ-calculus G-terms to the π-calculus in generic form, we may encoding them for some particular cases. For example, we may map the κ-calculus expression

$$\lfloor\,\rfloor\circ[\ !(\nu\,\kappa){}^{\dagger}\bar{m}_1(\tilde{\vec{x}})\,\check{\kappa}@[{}^{\dagger}\bar{m}_2]\,.\,P_1 \otimes !{}^{\dagger}\bar{m}_2(\tilde{\vec{x}})\,(\nu)@[{}^{\dagger}\bar{m}_2]\,.\,P_2 \otimes !{}^{\dagger}\bar{m}_1(\tilde{\vec{x}})\,(\nu)@\varnothing\,.\,P_3\ ]$$

to the polar π-calculus expression

$$({}^{\dagger}\bar{m}_1(\tilde{\vec{x}}).\,(\nu\,\kappa_1)(G_1 \,|\, [\![P_1]\!]_{p\pi}\{^{\kappa_1}/_{\kappa}\}) + {}^{\dagger}\bar{m}_2(\tilde{\vec{x}}).\,(G'' \,|\, [\![P_2]\!]_{p\pi})) \,|\, !{}^{\dagger}\bar{m}_3(\tilde{\vec{x}}).\,[\![P_3]\!]_{p\pi}$$

where $G_1 \overset{\text{def}}{=} {}^{\dagger}\bar{m}_1(\tilde{\vec{x}}).\,(\nu\,\kappa_2)(G_2 \,|\, [\![P_1]\!]_{p\pi}\{^{\kappa_2}/_{\kappa}\}) + \kappa_1\,.\,{}^{\dagger}\bar{m}_2(\tilde{\vec{x}}).\,(G'' \,|\, [\![P_2]\!]_{p\pi}),$

$G_i \overset{\text{def}}{=} {}^{\dagger}\bar{m}_1(\tilde{\vec{x}}).\,(\nu\,\kappa_{i+1})(G_{i+1} \,|\, [\![P_1]\!]_{p\pi}\{^{\kappa_{i+1}}/_{\kappa}\}) + \kappa_i\,.\,G_{i-1},$

$G'' \overset{\text{def}}{=} !{}^{\dagger}\bar{m}_1(\tilde{\vec{x}}).\,(\nu\,\kappa)[\![P_1]\!]_{p\pi}.$

10 Mapping between the Algebra of Exclusion and the κ-calculus

Expressions of the algebra of exclusion in [Noble00] has the following syntax:

$$e ::= 0 \,|\, m \,|\, e_1 \times e_2 \,|\, e_1 \,|\, e_2 \,|\, \bar{e}$$

where 0 is an empty expression; m represents the name of an object method, which indicates no exclusive relation when appears alone; expression $e_1 \,|\, e_2$ combines the exclusive relations described in e_1 and e_2 together without introducing any new exclusion; the expression $e_1 \times e_2$ is similar to $e_1 \,|\, e_2$ except an exclusive relation is added between every pair of a method name in e_1 and a method name in e_2; and the expression $\bar{e}$ is in fact the abbreviation for $e \times e$.

It is not the right word to say "encoding" the algebra of exclusion in the κ-calculus, since there is not dynamic semantics in the former, and the exclusion semantics can only be presented as a component of a term in the latter. However, we may still find some form of corresponding between them, with the help of the notion of the canonical higher-ordered GEC choice term, and the auxiliary function $n(e)$ which gives the set of all names in e. Let auxiliary function $n(e)$ represents the set of all names in e, η represent the unknown body of the method m, and η_i the unknown body of m_i, then the mapping from the exclusion algebra's expressions to the κ-calculus G-terms may look like:

$$\begin{aligned}
[\![0]\!]_\kappa &\stackrel{\text{def}}{=} \mathbf{0_G}\\
[\![m]\!]_\kappa &\stackrel{\text{def}}{=} «\eta» \,(!(\nu\,\kappa)\,{}^{\dagger}m(\tilde{x})\,\check{\kappa}@\emptyset.\,\eta\,)\\
[\![e_1 \mid e_2]\!]_\kappa &\stackrel{\text{def}}{=} [\![e_1]\!]_\kappa \boxplus [\![e_2]\!]_\kappa\\
[\![e_1 \times e_2]\!]_\kappa &\stackrel{\text{def}}{=} [\![e_1]\!]_\kappa \boxplus [\![e_2]\!]_\kappa \boxplus (\textstyle\bigotimes_{m\in n(e_1)} «\tilde{\eta}» !(\nu\,\kappa)\,{}^{\dagger}m(\tilde{x})\check{\kappa}@[\{{}^{\dagger}n\in n(e_2)\}].\eta) \boxplus (\textstyle\bigotimes_{m\in n(e_2)} «\tilde{\eta}» !(\nu\,\kappa)\,{}^{\dagger}m(\tilde{x})\check{\kappa}@[\{{}^{\dagger}n\in n(e_1)\}].\eta)
\end{aligned}$$

It is easy to see

$$\begin{aligned}
[\![m_1 \times m_2]\!]_\kappa &\equiv («\eta_1» \;!(\nu\,\kappa)\,{}^{\dagger}m_1(\tilde{x})\check{\kappa}@[{}^{\dagger}m_2].\,\eta_1) \;\boxplus\; («\eta_2» \;!(\nu\,\kappa)\,{}^{\dagger}m_2(\tilde{x})\check{\kappa}@[\,{}^{\dagger}m_1].\,\eta_2)\\
[\![\overline{m}]\!]_\kappa &\equiv [\![m \times m]\!]_\kappa\\
&\equiv «\eta» \,(!(\nu\,\kappa)\,{}^{\dagger}m(\tilde{x})\,\check{\kappa}@[{}^{\dagger}m].\,\eta)\\
[\![e_1 \times (\,e_2 \mid e_3)]\!]_\kappa &\equiv [\![(e_1 \times e_2) \mid (e_1 \times e_3)]\!]_\kappa\\
&\equiv [\![e_1 \times e_2]\!]_\kappa \boxplus [\![e_1 \times e_3]\!]_\kappa\\
[\![m_1 \times m_2 \times \ldots \times m_n]\!]_\kappa &\equiv [\![m_1 \times m_2]\!]_\kappa \boxplus \ldots \boxplus [\![m_1 \times m_n]\!]_\kappa \boxplus [\![m_2 \times \ldots \times m_n]\!]_\kappa\\
&\equiv «\tilde{\eta}» \textstyle\bigotimes_{i\in\{1\ldots n\}} !(\nu\,\kappa)\,{}^{\dagger}m_i(\tilde{x})\check{\kappa}@[\{{}^{\dagger}m_{j\in\{1\ldots n\}-\{i\}}\}].\eta_i
\end{aligned}$$

The directly mapping from the κ-calculus to the exclusion algebra is not possible since the exclusion relation is symmetric in the latter but asymmetric in the former. To enable a mapping we need an asymmetric version of the exclusion algebra: let the expression "$m_1 \times m_2$" only prevent the invocation of m_2 when method m_1 is running, but give no restriction on whether m_1 can start when m_2 is executing. Then we can have asymmetric exclusion algebra expressions:

$$\begin{aligned}
[\![\mathbf{0_G}]\!]_e &\stackrel{\text{def}}{=} 0\\
[\![«\eta»!(\nu\,\kappa)\,{}^{\dagger}m(\tilde{x})\,\check{\kappa}@\emptyset.\,\eta]\!]_e &\stackrel{\text{def}}{=} m\\
[\![«\eta»!(\nu\,\kappa)\,{}^{\dagger}m_1(\tilde{x})\,\check{\kappa}@[m_2].\eta]\!]_e &\stackrel{\text{def}}{=} m_1 \times m_2\\
[\![«\eta»!(\nu\,\kappa)\,{}^{\dagger}m(\tilde{x})\check{\kappa}@[{}^{\dagger}m_1,{}^{\dagger}m_2,\ldots,{}^{\dagger}m_n].\eta]\!]_e &\stackrel{\text{def}}{=} m \times (m_1 \mid m_2 \mid \ldots \mid m_n)\\
[\![\mathcal{G}_1 \boxplus \mathcal{G}_2]\!]_e &\stackrel{\text{def}}{=} [\![\mathcal{G}_1]\!]_e \mid [\![\mathcal{G}_2]\!]_e
\end{aligned}$$

11 Discussion and conclusion

The purpose of the κ-calculus is to provide a mathematical tool to model compositional concurrent object. In this calculuse, the locking/unlocking becomes primitive, and locking can be initialised. These make some issues special from those in normal π-liked calculi.

11.1 Unification of product and sum and others

From the result of section 5 and 6, we can derive more useful properties.

Lemma 11−123: If $\forall_{i,j\in I}.(\hat{\kappa}\notin fon(P_i) \wedge {}^{\dagger}m_i \in J_j)$ then the κ-calculus term $\lfloor\,\rfloor\circ\lceil(\nu\,\kappa)\bigotimes_{i\in I} !{}^{\dagger}m_i(\tilde{x})\check{\kappa}@J_i.P_i\rfloor$ is equivelant to the π-calculus term $\sum_{i\in I} {}^{\dagger}m_i(\tilde{x}).P_i$.

Proof: Both can and only can commit on the same set of input actions. For an arbitrary action ${}^{\dagger}m_d(\tilde{u})$ where $d\in I$, notice that $\forall_{i\in I}.({}^{\dagger}m_i \in J_d)$, then by rules tr_IN and str-SCP3, the former can commit on ${}^{\dagger}m_d(\tilde{u})$ and reduce to $P_d\{\tilde{u}/\tilde{x}\} \mid (\nu\,\kappa)\lfloor\,\check{\kappa}@J_d\rfloor\circ\lceil\bigotimes_{i\in I}!{}^{\dagger}m_i(\tilde{x})\check{\kappa}@J_i.P_i\rfloor \equiv P_d\{\tilde{u}/\tilde{x}\}$, and by the reduction rules of the π-calculus, the latter can also commit on ${}^{\dagger}m_d(\tilde{u})$ and reduce to $P_d\{\tilde{u}/\tilde{x}\}$. ■

Lemma 11−124: If $\forall_{i\in I}.(excl(B_i)=\emptyset \vee (excl(B_i) = \{guard(B_i)\} \wedge \forall_{j\in I,\, j\neq i}.(guard(B_i)\neq guard(B_j))))$, that is, none of branches can fire a locker locking another branch, then $\lfloor\,\rfloor\circ\lceil\bigotimes_{i\in I} B_i\rfloor \sim_g \prod_{i\in I}(\lfloor\,\rfloor\circ\lceil B_i\rfloor)$.

Proof: It is easy to be verified from the definition of $\sim_g$. ■

The above two corollaries indicate that for input guarder processes, the π-calculus operator '+' and '$\mid$' are related in the κ-calculus: they are merely two extrame cases of the '⊗' operator. In order to simplify our κ-calculus expressions, we introduce the following abbreviations

Notation 11−125: We define the process abbreviations $\overset{+}{m}(\tilde{x}).P \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil\, !\overset{+}{m}(\tilde{x})\,(\nu)@[\overset{+}{m}].P\rceil$;

$$\textstyle\sum_{i\in I}\overset{+}{m}_i(\tilde{x}).P_i \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil \bigotimes_{i\in I}!\overset{+}{m}_i(\tilde{x})\,(\nu)@\overset{+}{\mathcal{N}}.P_i\rceil;$$

$$!\overset{+}{m}(\tilde{x}).P \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil !\overset{+}{m}(\tilde{x})(\nu)@\varnothing.P\rceil.$$

From the easy-to-be proven facts $\prod_{i\in I}\lfloor\,\rfloor\circ\lceil !\overset{+}{m}_i(\tilde{x})\,(\nu)@[\overset{+}{m}_i].P_i\rceil \equiv \lfloor\,\rfloor\circ\lceil \bigotimes_{i\in I}!\overset{+}{m}_i(\tilde{x})\,(\nu)@[\overset{+}{m}_i].P_i\rceil$ and $\prod_{i\in I}\lfloor\,\rfloor\circ\lceil !\overset{+}{m}_i(\tilde{x})\,(\nu)@\varnothing.P_i\rceil \equiv \lfloor\,\rfloor\circ\lceil \bigotimes_{i\in I}!\overset{+}{m}_i(\tilde{x})\,(\nu)@\varnothing.P_i\rceil$

we can also introduce the abbreviations $\prod_{i\in I}\overset{+}{m}_i(\tilde{x}).P_i \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil \bigotimes_{i\in I}!\overset{+}{m}_i(\tilde{x})\,(\nu)@[\overset{+}{m}_i].P_i\rceil$ and $\prod_{i\in I}!\overset{+}{m}_i(\tilde{x}).P_i \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil \bigotimes_{i\in I}!\overset{+}{m}_i(\tilde{x})\,(\nu)@\varnothing.P_i$.

In other words, for input-guarded terms in the π-calculus, the parallel composition, mutual exlcusive choice, replication, etc. are all special cases of *GEC* terms in the κ-calculus.

11.2 ω-receptiveness and linear receptiveness

[Sangiorgi97] using types to distinguish the input port of communication channels as either ω or linear receptiveness. In our κ-calculus, they are two special cases for generic receivers of *GEC*-terms:

ω-receptiveness $\quad !\overset{+}{m}(\tilde{u}).P \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil\, !\overset{+}{m}(\tilde{x})(\nu)@\varnothing.P\rceil$;
linear-receptiveness $\quad \overset{+}{m}(\tilde{u}).P \overset{\text{def}}{=} \lfloor\,\rfloor\circ\lceil\, !\overset{+}{m}(\tilde{x})(\nu)@[\overset{+}{m}].P\rceil$.

When used for message forwarding, input ports with different receptiveness will behaviour differently. For channel *m*,

With ω-receptiveness, $\quad (\nu\, n)(\,!\overset{+}{m}(\tilde{x}).\overline{n}\langle\tilde{x}\rangle \mid \lfloor \check{\kappa}.@[\overset{+}{n}]\rfloor\circ\lceil\, !\overset{+}{n}(\tilde{x})L.P\rceil) \approx_r \lfloor \check{\kappa}.@[\overset{+}{m}]\rfloor\circ\lceil\, !\overset{+}{m}(\tilde{x})L.P\rceil)$;
With linear-receptiveness, $\quad (\nu\, n)(\,\overset{+}{m}(\tilde{x}).\overline{n}\langle\tilde{x}\rangle \mid \lfloor \check{\kappa}.@[\overset{+}{n}]\rfloor\circ\lceil\, !\overset{+}{n}(\tilde{x})L.P\rceil) \not\approx_r \lfloor \check{\kappa}.@[\overset{+}{m}]\rfloor\circ\lceil\, !\overset{+}{m}(\tilde{x})L.P\rceil)$,
but $\quad (\nu\, n)(\,\overset{+}{m}(\tilde{x}).\overline{n}\langle\tilde{x}\rangle \mid \lfloor \check{\kappa}.@[\overset{+}{n}]\rfloor\circ\lceil\, !\overset{+}{n}(\tilde{x})L.P\rceil) \approx_r \lfloor \check{\kappa}.@[\overset{+}{m}]\rfloor\circ\lceil\, !\overset{+}{m}(\tilde{x})(\nu)@[\overset{+}{m}].P\rceil)$.

The κ-calculus explanation for this difference is, according to our definition above, the ω-receiver has an empty control while linear-receiver has not (but an un-releasable locking control). What is the connection between the generic receptiveness types and controls which are neither empty nor un-releasable will be an interesting topic to be studied in the future work.

11.3 Compositional object model

A more sophisticated model involves a set of more communication synchronisation channels and keeps the unlocking signal channel within the control componants. Let process *F* reprsent the functionality of an object:

$$F \overset{\text{def}}{=} (\tilde{m})\lfloor\,\rfloor\circ\lceil \bigotimes_{i\in I}!\overset{+}{m}_i(\overline{s}_m,\overline{s}_f,\overline{r}_m,\overline{t}_m,\tilde{x})(\nu)@\varnothing.M_i\langle\tilde{m},\overline{s}_m,\overline{s}_f,\overline{r}_m,\overline{t}_m,\tilde{x}\rangle\rceil \qquad 11-1$$
$$\equiv (\tilde{m})\textstyle\prod_{i\in I}!\overset{+}{m}_i(\overline{s}_m,\overline{s}_f,\overline{r}_m,\overline{t}_m,\tilde{x}).M_i\langle\tilde{m},\overline{s}_m,\overline{s}_f,\overline{r}_m,\overline{t}_m,\tilde{x}\rangle$$

Here each $\overset{+}{m}_i$ refers to a method, $\tilde{x}$ the arguaments to the method call, $\overline{s}_m$ acknowledges the receiving of the call, $\overline{s}_f$ indicates the start of method body execution, $\overline{r}_m$ is the link to the required return value, and $\overline{t}_m$ signals the termination of method body execution. Note, some abbreviation introdued earlies in this section has been used in the latter part of the above equation. With another previously defined abbreviation $\overline{t}.P \overset{\text{def}}{=} \overline{t}\mid P$, then the generic form of M_i may look like

$$M_i \overset{\text{def}}{=} (\tilde{m},\overline{s}_m,\overline{s}_f,\overline{r}_m,\overline{t}_m,\tilde{x})\ (\overline{s}_m \mid \overline{s}_f \mid \mathbb{P}_i«(\tilde{u})(\overline{r}_m\langle\tilde{u}\rangle \mid \mathbb{P}'_i«\overset{+}{t}_m.\mathbf{0}»)»)$$
$$\equiv (\tilde{m},\overline{s}_m,\overline{s}_f,\overline{r}_m,\overline{t}_m,\tilde{x})\ \overline{s}_m.\ \overline{s}_f.\ \mathbb{P}_i«(\tilde{u})(\overline{r}_m\langle\tilde{u}\rangle \mid \mathbb{P}'_i«\overset{+}{t}_m.\mathbf{0}»)»$$

where, $\overline{r}_m\langle\tilde{u}\rangle \mid \mathbb{P}'_i«\overset{+}{t}_m.\mathbf{0}»$, the continuation of $\mathbb{P}_i$, indicates that the requested value may be obtained and immediately returned via $\overline{r}_m$ in the middle of the excution (called early return).

For the generic form of the object functionality *F* in the equation 11−1, assume $\tilde{m}{:}\tilde{\overline{\lambda}}_m, \tilde{x}{:}\tilde{\overline{\lambda}}_x, \overline{s}_m{:}\overline{\lambda}_s, \overline{s}_f{:}\overline{\lambda}_f, \overline{r}_m{:}\overline{\lambda}_r, \overline{t}_m{:}\overline{\lambda}_t$, then $M_i{:}\,\mathsf{pabs}(\tilde{\overline{\lambda}}_m,\overline{\lambda}_s,\overline{\lambda}_f,\overline{\lambda}_r,\overline{\lambda}_t,\tilde{\overline{\lambda}}_x)$. Let $\xi_i \overset{\text{def}}{=} \mathsf{pabs}(\overline{\lambda}_s,\overline{\lambda}_f,\overline{\lambda}_r,\overline{\lambda}_t,\tilde{\overline{\lambda}}_x)$,then we may define an empty control $\mathbb{E}{:}\,\mathsf{gabs}(\tilde{\overline{\lambda}}_m,\tilde{\xi})$ for *F* such that

$$F \overset{\text{def}}{=} (\tilde{m})\lfloor\,\rfloor\circ[\bar{\bar{\mathbb{E}}}\langle\overset{+}{\tilde{m}}\rangle \langle\!\langle M_{i\in I}\langle\tilde{m}\rangle\rangle\!\rangle] \qquad 11\text{–}2$$

$$\bar{\bar{\mathbb{E}}} \overset{\text{def}}{=} (\overset{+}{\tilde{m}})\langle\!\langle\tilde{\eta}\rangle\!\rangle \bigotimes_{i\in I}\overset{+}{m}_i(\bar{s}_m,\bar{s}_f,\bar{r}_m,\bar{t}_m,\tilde{\bar{x}}).\eta_i\langle\bar{s}_m,\bar{s}_f,\bar{r}_m,\bar{t}_m,\tilde{\bar{x}}\rangle \qquad 11\text{–}3$$

and both $\bar{\bar{\mathbb{E}}}$ and F are well-typed. $\bar{\bar{\mathbb{E}}}$ is called as an empty control because it does nothing else rather than passes messages to corresponding mathod bodies. However, we may conside that $\bar{\bar{\mathbb{E}}}$ provides the interface, a set of the input polar of channels (methods), to the object. While $\bar{\bar{\mathbb{E}}}$ is presented in the form of $\bar{\bar{\mathbb{E}}}\equiv\bar{\bar{\mathbb{E}}}_1\boxplus\bar{\bar{\mathbb{E}}}_2\boxplus\ldots\boxplus\bar{\bar{\mathbb{E}}}_n$, we may consider that each $\bar{\bar{\mathbb{E}}}_i$ describes a portion of the interface. For the generic form of objects, the functionality $F:\mathsf{pabs}(\overset{\pm}{\tilde{\lambda}}_m)$, is a process with an interface, an empty exclusion control, and a set of method body definitions.

Generally, an object can be modelled as a process with a set of input polars, representing methods, as the only interface for communication. Certainly, there may also exist another set of links representing global knowldges within the environment, but we may hide them by assume they are always included in the method parameters by default.

Let C be the control process which represents the concurrent behaviours, and itself can be divided into several parts, $\lfloor\,\rfloor$ the empty locking list, $\mathbb{G}$ describes the exclusion and $\{G_{T\,i\in I}\}$ controls timing, such as early return, etc.

$$\bar{\mathbb{G}} \overset{\text{def}}{=} (\overset{+}{\tilde{n}})\langle\!\langle\tilde{\eta}\rangle\!\rangle \bigotimes_{i\in I} !(\nu\,\kappa)\,\overset{+}{n}_i(\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{x}})\,\check{\kappa}@J_i.\ \eta_i\langle\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{x}},\hat{\kappa}\rangle \qquad 11\text{–}4$$

$$C \overset{\text{def}}{=} (\tilde{m},\overset{+}{\tilde{n}})\ \lfloor\,\rfloor\circ[\bar{\mathbb{G}}\langle\overset{+}{\tilde{n}}\rangle \langle\!\langle G_{T\,i\in I}\langle\tilde{m}\rangle\rangle\!\rangle] \qquad 11\text{–}5$$

where $\tilde{n}:\overset{\pm}{\tilde{\lambda}}_m, \bar{s}_n:\bar{\lambda}_s, \bar{r}_n:\bar{\lambda}_r, \bar{t}_n:\bar{\lambda}_t, \kappa:\overset{\pm}{\delta}$ and $\bar{\mathbb{G}}:\mathsf{gabs}(\tilde{\bar{\lambda}}_m,\tilde{\xi}_c)$, with $\xi_{ci}\overset{\text{def}}{=}\mathsf{pabs}(\bar{\lambda}_s,\bar{\lambda}_f,\bar{\lambda}_r,\bar{\lambda}_t,\tilde{\bar{\lambda}}_x,\bar{\delta})$. An example of G_{Ti} can be

$$\begin{aligned} G_{Ti} &\overset{\text{def}}{=} (\tilde{m},\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{x}},\hat{\kappa})\ (\nu\ s_m,r_m,t_m)(\bar{s}_n \mid \bar{m}_i\langle\bar{s}_m,\bar{s}_f,\bar{r}_m,\bar{t}_m,\tilde{\bar{x}}\rangle \mid \overset{+}{s}_m.\overset{+}{r}_m(\tilde{y}).(\bar{r}_n\langle\tilde{y}\rangle \mid \overset{+}{t}_m.(\bar{t}_n \mid \hat{\kappa}))) \\ &\equiv (\tilde{m},\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{x}},\hat{\kappa})\ (\nu\ s_m,r_m,t_m)\ \bar{s}_n.\bar{m}_i\langle\bar{s}_m,\bar{s}_f,\bar{r}_m,\bar{t}_m,\tilde{\bar{x}}\rangle.\overset{+}{s}_m.\overset{+}{r}_m(\tilde{y}).\bar{r}_n\langle\tilde{y}\rangle.\overset{+}{t}_m.\bar{t}_n.\hat{\kappa}. \end{aligned}$$

Note, in this model the signal channel κ is completely encapsulated within the control C, and the functional object F needs no knowledge about it at all. Now, write

$$\begin{aligned} M'_i &\overset{\text{def}}{=} (\tilde{m},\bar{s}_m,\bar{s}_f,\bar{r}_m,\bar{t}_m,\tilde{\bar{x}})(\bar{s}_m \mid \bar{s}_f \mid \mathbb{P}_i\langle\!\langle(\tilde{u})(\bar{r}_m\langle\tilde{u}\rangle \mid \mathbb{P}'_i\langle\!\langle\overset{+}{t}_m.\hat{\kappa}\rangle\!\rangle)\rangle\!\rangle)\rangle\!\rangle) \qquad 11\text{–}6 \\ &\equiv (\tilde{m},\bar{s}_m,\bar{s}_f,\bar{r}_m,\bar{t}_m,\tilde{\bar{x}})\bar{s}_m.\bar{s}_f.\mathbb{P}_i\langle\!\langle(\tilde{u})(\bar{r}_m\langle\tilde{u}\rangle \mid \mathbb{P}'_i\langle\!\langle\overset{+}{t}_m.\hat{\kappa}\rangle\!\rangle)\rangle\!\rangle \end{aligned}$$

then we have

$$(\tilde{n})\,(\nu\ \tilde{m})\,(F\langle\tilde{m}\rangle \mid C\langle\tilde{m},\overset{+}{\tilde{n}}\rangle)\ \approx_{\kappa r}\ (\tilde{n})\ \lfloor\,\rfloor\circ[\bar{\mathbb{G}}\langle\overset{+}{\tilde{n}}\rangle\langle\!\langle M'_{i\in I}\langle\overset{+}{\tilde{n}}\rangle\rangle\!\rangle]. \qquad 11\text{–}7$$

And $(\overset{+}{\tilde{n}})\lfloor\,\rfloor\circ[\bar{\mathbb{G}}\langle\overset{+}{\tilde{n}}\rangle\ \langle\!\langle\tilde{M}'\rangle\!\rangle]$, the composition of F and C, is an object process.

With this model, the structure of the control C itself actually has already separated some different aspects of concurrency: The locking list Λ, can be viewed as a thread monitor, and may be extended for access controlling or other usage; The canonical higher-ordered GEC choice $\bar{\mathbb{G}}$ describes exclusion relation among the methods; and the tail controller $\{G_{Ti\in I}\}$ acts like a scheduler co-ordinating some timing. While $\bar{\mathbb{G}}$ is presented in the form $\bar{\mathbb{G}}\equiv\bar{\mathbb{G}}_1\boxplus\bar{\mathbb{G}}_2\boxplus\ldots\boxplus\bar{\mathbb{G}}_n$, then each indicates a portion of the interface and the exclusion relation among the methods within this portion.

Several control processes may be compounded to from a new control process with the composite concurrent behaviours. For example, we may construct a control proccess as $C\overset{\text{def}}{=}(\tilde{m},\overset{+}{\tilde{n}})\ (\nu\ \tilde{p})(\ C_1\langle\tilde{p},\overset{+}{\tilde{n}}\rangle \mid C_2\langle\tilde{m},\overset{+}{\tilde{p}}\rangle)$.

Corrsponding to the object model, a method call which waits the return value can be modelled as :

$$(\nu\ s_n,s_f,r_n,t_n)(\ \bar{n}\langle\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{x}}\rangle \mid \overset{+}{r}_n(\tilde{y}).\overset{+}{t}_n.Q\langle\tilde{y}\rangle\ ) \qquad 11\text{–}8$$

where $\tilde{\bar{y}}$ are the values returned from the method call, and Q is the continuation process which waiting the return value. When no return value needed, a proceduel call can be modelled as:

$$(\nu\ s_n,s_f,r_n,t_n)(\ \bar{n}\langle\bar{s}_n,\bar{s}_f,\bar{r}_n,\bar{t}_n,\tilde{\bar{x}}\rangle \mid \overset{+}{t}_n.Q\ ) \qquad 11\text{–}9$$

Now the concurrent behaviour of the method calls can be controlled by the control processes in the object side. For example, with the G_{Ti} above, the process Q in equation 11–8 or 11–9 will have to wait the termination of the called method body before it can start to execute. However, if we swap the order of $\overset{+}{t}_m$ and $\bar{t}_n$ in G_{Ti}, then Q will be able to executing concurrently with the method body without waiting.

This object model clearly demonstrates that the κ-calculus provides a good platform to separate different aspects in modelling concuttent objests. These separations we can easily achieved include

1. Separate the functionality F (or method bodies) from the concurrent behaviours C;

2. Separate the current locking status Λ from the specification of concurrency controls, $\overline{\mathbb{G}}$ and G_{Ti}.
3. Separate the specification of exclusion policy $\overline{\mathbb{G}}$ form the specification of synchonisation timing G_{Ti};
4. Separate several concurrent behaviours into some different control processes, then compose them together.

More detailed study on compositional concurrent objects modelling, behaviours separation and their properties will be carried out in [Zhang02C] and [Zhang02D].

11.4 Process Equivalence verse Object Equivalence verse Abstracted Object Equivalence

In this paper we have met three different levels of behaviours equivalence: process bisimulations, object equivalences (G-bisimulataions) and exclusion equivalence ($\mathbb{G}$-bisimulation).

The process bisimulations, a kind of process equivalence, play the same role in the κ-calculus as that in convenience π-calculi. They are used to describe the behaviour similarity of processes at certain statues of the evalution.

The G-bisimulations, describe the equivalence between GEC choice terms, are defined via process equivalences. However, the G-bisimulations remove the evolution status from the behavious comparason, and therefore can be egarded as the comparason of the definition of object behavious. Object processes constructed from GEC choice terms which satisfy some G-bisimulation will guarantee the corresponding process bisimulation if start with the same initial status, and therefore we may consider G-bisimulations as *Object Equivalences*,

The $\mathbb{G}$-bisimulation further removes all other behaviours except: 1. The object interface; 2. Exclusion relations among the methods of the object. We may consider that the $\mathbb{G}$-bisimulation providers an equivalence measurement between abstracted concurrent objects.

11.5 Other future works

The term Λ which records the locking status is in fact a monitor. Its separation from the definition of exclusion policies does not only allow exclusion behaviours to be specified in a clear and pure form, but also enable us to define the "monitoring" policies separately. For example, we may check the "threads' family tree" (thread creation history) via key κ in Λ to solve the method call-back problem, or, check the ownership in Λ to solve the access priviege or trustedagent problem.

Thread ID history The word "call-back" refers to a thread which is executing some method of an object calls another a method of the same object again. However, some locking mechanism may have been triggered by this thread in the earlier access. To avoid deadlock, the principle that "a thread should never lock itself" is widely adopted. This principle is actually always used by sequencial progamming. The "sequencial progamming" is a misleading term since in fact it is equivelant to concurrent programming with single thread restriction.

The problem of "a thread should never lock itself" is that, when a thread which is accessing some objects splits, how should the locking mechanism should re-act if more than one child threads try to make self-call or call-back?

One solution, we proposed, is the "thread ID with historical information" mechanism. That is, whenever a thread splits, each of the split branches becomes a child thread and is given a new ID which is not only used to distinguish individual threads, but also contains the information of all its parents' ID. When a thread triggers a lock in an object, it registers its ID in the lock's owner list, and will de-register when leaving. When a thread attempts to access a locked object, its ID will be checked against the lock's owners-list. The thread will not be blocked if the lock's owners-list contains no thread ID other than this thread's own ID or IDs of some its ancestor, but will be blocked by a lock owned by a thread from a different branch of the same family tree. A lock is removed when and only when its owner list is empty.

The extension for allowing such a threads' ID checking mechanism is left to the future works.

Allow locking keys to be transmitted The κ-calculus distinguishes communication polars and locking signals. As pointed out in section 2, this distinction can prevent cross using between them, for example, sending a message to some object should not release a lock at somewhere else. For the current version of κ-calculus, this distinction is also used to syntactically prohibt a key to be transmitted in communicataion.

Since our purpose of using the κ-calculus is to model compositional concurrent objects, where concurrency controls can be completely separated from functionality, the prohibition of transmitting locking key can help us to guarantee a privately defined key never cross the boundary of the control process.

However, in some applications, one may do want to transmit a release key. For example, in a group working project, one of the group member who has locked some object may want transfer the control of this object to another member of the group before exit the access. In such a case, the prohibition of transmitting locking key can be removed from the syntax of the κ-calculus. We leave this to the future work.

Selective unlock Sometimes when release a lock which blocks a set of methods, we may not want all of these methods to be released all at once, but just a subset of it. One of the application of this, can be access priviledge control. To enable the selective unlocking, the unlock signal may have a form like $\hat{\kappa}@J$, where J is the set of methods to be released. A lock is removed only when the set of methods which remain as locked become empty. The study of selective unlock is also left for future work.

References:

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

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

[Amadio97] Roberto M. Amadio, "An Asynchronous Model of Locality, Failure, and Process Mobility", In D. Garlan and D. Le Metayer, editor, *Proceedings of The Second International. Conference on Coordination Models and Languages (COORDINATION'97)*, LNCS 1282, Springer, 1997

[Bos89] J. van den Bos and C. Laffra, "PROCOL A Parallel Object Language with Protocols, " in Proceedings of the 1989 OOPSLA Conference, New Orleans, Louisiana, September 1989.

[Bos91] Jan van den Bos, Chris Laffra: "PROCOL: A Concurrent Object-Oriented Language with Protocols Delegation and Constraints. " Acta Informatica 28(6): 511-538 (1991)

[Busi95] Nadia Busi and Roberto Gorrieri, "Distributed Conflicts in Communicating Systems", in Christine Mingins, Roger Duke and Bertrand Meyer, editors, *Object-Based Models and Languages for Concurrent Systems*, LNCS vol 924, pages 49-65, Springer-Verlag, 1995. URL: ftp://ftp.cs.unibo.it/pub/techreports/94-08.ps.gz

[Crno98] Lobel Crnogorac, Anand S. Rao, and Kotagiri Ramamohanarao, "Classifying Inheritance Mechanisms I concurrent Object-Oriented Programming," in Eric Jul, editor, *Proceedings of ECOOP'98*, volume 1445 of *Lecture Notes in computer Science*, pages 571-600. Springer Verlag, 1998.

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

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

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

[Honda95] Kohei Honda and Mario Tokoro, "On Reduction-based Process Semantics", *Theoretical Computer Science*, 152(2):437-486, 1995.

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

[Jalloul94] Ghinwa Jalloul, "*Concurrent Object-Oriented Systems: A Disciplined Approach*", PhD Dissertation, University of Technology, Sydney, Australia, June 1994

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

[Laff92] C. Laffra, "PROCOL: a Concurrent Object Language with Protocols, Delegation, Persistence and Constraints, " Ph.D. thesis, Erasmus Universiteit, Rotterdam, the Netherlands, May 1992.

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

[Matsuoka93] Satoshi Matsuoka, "Language Features for Reuse and Extensibility in Concurrent Object-Oriented Programming", PhD thesis, Department of Information Science, University of Tokyo, Japan, April 1993

[McHale94] Ciaran McHale, "Synchronisation in Concurrent, Object-oriented Languages: Expressive Power, Genericity and Inheritance", PhD. Thesis, Department of Computer Science, Trinity college, University of Dublin, Ireland, October 1994. URL: ftp://ftp.dsg.cs.tcd.ie/pub/doc/dsg_86.ps.gz

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

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

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

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

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

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

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

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

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

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

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

[Pierce93] Benjamin C. Pierce and Davide Sangiorgi, "Typing and Subtyping for Mobile Processes", In *Proceedings of 8th Symposium on Logic in Computer Science*, pages 409-454, IEEE Computer society Press, 1993. URL: http://www.inria.fr/meije/personnel /Davide.Sangiorgi/mypapers.html

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

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

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

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

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

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

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

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

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

[Schneider97] Jean-guy Schneider and Markus Lumpe, "Synchronizing Concurrent Objects in the ", Proceedings of Langages et Modèles à Objets '97, Roland Ducournau and Serge Garlatti (Ed.), Hermes, Roscoff, October 1997, pp. 61-76. URL: ftp://ftp.iam.unibe.ch /pub/scg/Papers/lmo97.ps.gz

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

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

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

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

[Zhang02A] Xiaogang Zhang and John Potter, "*The Responsive Bisimulations in the polar π-calculus*", Technical report UNSW-CSE-TR-0203.

[Zhang02B] Xiaogang Zhang and John Potter, "*On Responsive Bisimulations in the κ-calculus*", Technical report UNSW-CSE-TR-0205.

[Zhang02C] Xiaogang Zhang and John Potter, "*Compositional Concurrent Objects*", in preparation.

[Zhang02D] Xiaogang Zhang and John Potter, "A *Compositional Concurrent Object Model, -- From Theory to Practise*", in preparation.