Semantics and Encoding of the kell-m Calculus

Rolando Blanco and Paulo Alencar
David R. Cheriton School of Computer Science
University of Waterloo

Technical Report CS-2011-01

Abstract

We present kell-m, an asynchronous higher-order process algebra with hierarchical localities. The main focus of this report is on the operational semantics and behavioural equivalences for kell-m. The operational semantics determine how systems represented using kell-m evolve; the behavioural equivalences determine what it means for two kell-m processes to behave similarly. We also present and encoding of kell-m into MMCπ, the variation of the π-calculus as implemented in the Mobility Model Checker (MMC).

1 Introduction

Kell-m is a process algebra where computations are modelled as processes executing in parallel. As it is traditional in process algebras, processes communicate with each other via *channels*. Channels are a kind of a more general construct, called *names*. A name is an identifier, and it is only useful for comparing for identity with other names [10].

In a communication on a particular channel one of the processes writes to the channel and another process reads from the channel. As part of the communication names and processes can be transmitted from the writer process to the reader process. Because processes can be transmitted in communications, kell-m is a *higher-order* process algebra.

Kell-m is derived from the Kell calculus [16, 15, 3]. The Kell calculus was created for studying component-based distributed programming. Built around a π-calculus core, in the Kell calculus processes execute in localities called *kells*, where kells are identified by names. The term *kell* was chosen by the creators of the Kell calculus in a loose analogy with biological cells. Kells can be dynamically created, stopped, deleted, and the process running within a kell can be replaced or composed with other processes.

Because kells are themselves processes, they can be localized within other kells, forming a tree containment hierarchy. In the traditional Kell calculus, direct communication among processes can occur only if the communicating processes are separated by at most one kell boundary. This is in contrast with kell-m, where we allow communication among processes independently of their kell location. Because of this deviation from the original Kell calculus, we name our calculus the *kell-m* calculus. By convention, we do not capitalize the *k* in *kell-m*, unless we are starting a sentence.

We start the presentation of kell-m with its syntax in Section 2. We formalize the operational semantics of kell-m in Section 3. In Section 4 we present kμ, a modal logic used for property specification of systems modelled using kell-m. An encoding of kell-m into MMCπ is presented in Section 5. MMCπ is an extension of the π-calculus as specified in [1, 17] and implemented in the Mobility Model Checker (MMC). An encoding for kμ is presented in Section 6. We conclude the report in Section 7.

2 Syntax

Every process in the kell-m calculus follows the syntax specified in Figure 1. P represents a kell-m process, **0** is the *null* process (a process that does not perform any actions), and $|$ is parallel composition of processes. A write action on channel a is specified $\overline{a}(\widetilde{w})$, where $\widetilde{w}$ is the sequence of names and processes being written on channel a. It is possible for a write action to transmit no values as part of the communication: $\overline{a}()$. For a channel a, when no values are passed

$$
\begin{array}{rcl}
P & ::= & \mathbf{0} \mid P|P \mid \overline{a}(\widetilde{w}) \mid \xi \triangleright P \mid \textbf{new}\ a\ P \mid K[P] \mid x \mid p(\widetilde{w}) \\
\xi & ::= & a(\widetilde{v}) \mid K[x] \\
v & ::= & c \mid x \\
w & ::= & c \mid P \\
p(\widetilde{c}) & \stackrel{def}{=} & P
\end{array}
$$

Figure 1: Kell-m Syntax

in the channel, we sometimes write $\overline{a}$ for $\overline{a}()$. Note the calculus is asynchronous: a write operation cannot be followed by a process.

A read action is specified using *triggers*. A trigger has the form $\xi \triangleright P$, where ξ is called a *pattern*, and P is the process to execute after the read. The pattern determines the channel on which the read will occur. For example, in $a(\widetilde{c}) \triangleright P$, the pattern $a(\widetilde{c})$ specifies a read on channel a. When the read expression in a trigger pattern is matched to a write expression, the result of the communication is the process specified in the trigger, P in previous the example, with the names in $\widetilde{c}$ replaced by the values passed by the writer process. Therefore,

$$\overline{a}(d) \mid (a(e) \triangleright P) \rightarrow P\{d/e\}$$

where $P\{d/e\}$ represents the process P after all occurrences of name e have been replaced by d. For a match to occur, the number of values passed in the write expression must match the number of values expected by the read expression.

The construct **new** $a\ P$ is called *name restriction*, and it is used to specify a private name a, to be used in P. For example, assume processes:

$$P_1 : \overline{a}(d) \qquad P_2 : a(c) \triangleright R \qquad P_3 : a(e) \triangleright Q$$

When composed in parallel, P_1 can communicate with either P_2 and P_3. To guarantee the communication happens only between P_1 and P_2, a restriction on name a for P_1 and P_2 can be specified:

$$\textbf{new}\ a\ (P_1 \mid P_2) \mid P_3$$

A private name is also called a *restricted name*. In the example, processes P_1 and P_2 are said to be within the scope of the restricted name a.

We write **new** *a,b,c* P to represent **new** a **new** b **new** $c\ P$, and **new** $\widetilde{c}\ P$ to represent **new** $c_1, c_2, \ \ldots, c_n\ P$ when $\widetilde{c} = c_1, c_2, \ldots, c_n$.

Processes can execute inside *kells*, where a kell is just another kind of name (recall channels are also names). $K[P]$, specifies a process P running inside kell K. When $K[P]$, we say that process P is *located* in kell K. For a given process, we refer to its *kell structure* as the containment relationships between kells in the process.

There is no restriction in the use of uppercase or lowercase letters; we typically use uppercase for kell names and processes, and lowercase for channels and process variables.

Trigger patterns can also specify kells. For example, $K[x] \triangleright R$ matches kell $K[P]$. The process variable x in R is replaced by P after the match:

$$(K[x] \triangleright R) \mid K[P] \rightarrow R\{P/x\}$$

When a kell is matched by a trigger, we say that the kell is *passivated*. Passivation can be used to alter the process running within a kell:

$$(K[x] \triangleright K[R|x]) \mid K[P] \rightarrow K[R\{P/x\}|P]$$

Passivation can also be used to move a process from one location to another:

$$T[P \mid K[Q]] \mid L[R \mid K[x] \triangleright K[x]] \rightarrow T[P] \mid L[R \mid K[Q]]$$

In the previous process, $K[P]$ was moved from kell T to kell L.

$p(\widetilde{w})$, represents a *process invocation*, where p is the name of the process, and $\widetilde{w}$ are the invocation parameters. The process must have been previously defined: $p(\widetilde{c}) \stackrel{def}{=} P$. An invocation $p(\widetilde{w})$ is equivalent to $P\{\widetilde{w}/\widetilde{c}\}$.

Since triggers are consumed when there is a match between a write action and the read action specified in the trigger pattern, *recurrent triggers* are introduced. A recurrent trigger is specified using $\diamond$ instead of $\triangleright$. For example:

$$(a(c) \diamond P) \mid \overline{a}(d) \rightarrow (a(c) \diamond P) \mid P\{d/c\}$$

In the previous example, the process $\overline{a}(d)$ is matched with the pattern specified by the recurrent trigger $a(c) \diamond P$. The resulting process after the match is $P\{d/c\}$, and the trigger is not consumed. Hence, the trigger can be matched to any number of write operations on channel a.

Similarly to other algebras derived from the Kell calculus, recurrent triggers are derived expressions in kell-m, since they can be expressed in terms of regular triggers [16, 15, 3]. For triggers specifying reads on channels, $\diamond$ is defined as the following *fixed point* [16]:

$$a(\widetilde{c}) \diamond P \equiv \textbf{new}\ t\ (Y(a, \widetilde{c}, P, t) \mid \overline{t}(Y(a, \widetilde{c}, P, t)))$$

with,

$$Y(a, \widetilde{c}, P, t) \stackrel{def}{=} t(y) \triangleright (a(\widetilde{c}) \triangleright (P \mid y \mid \overline{t}(y)))$$

Hence,

$$\begin{array}{rcl} a(\widetilde{c}) \diamond P & \equiv & \textbf{new}\ t\ (Y(a, \widetilde{c}, P, t) \mid \overline{t}(Y(a, \widetilde{c}, P, t))) \\ & \equiv & \textbf{new}\ t\ (t(y) \triangleright (a(c) \triangleright (P \mid y \mid \overline{t}(y))) \mid \overline{t}(Y(a, \widetilde{c}, P, t))) \\ & \rightarrow & a(\widetilde{c}) \triangleright (P \mid Y(a, \widetilde{c}, t) \mid \overline{t}(Y(a, \widetilde{c}, P, t))) \end{array}$$

If $\overline{a}(\widetilde{d}) \mid a(\widetilde{c}) \triangleright (P \mid Y(a, \widetilde{c}, P, t) \mid \overline{t}(Y(a, \widetilde{c}, P, t)))$, we obtain:

$$P\{\widetilde{d}/\widetilde{c}\} \mid Y(a, \widetilde{c}, P, t) \mid \overline{t}(Y(a, \widetilde{c}, P, t))) \equiv P\{\widetilde{d}/\widetilde{c}\} \mid (a(\widetilde{c}) \diamond P)$$

Similarly, for trigger patterns specifying kells:

$$K[X] \diamond P \equiv \textbf{new}\ t\ (Y_k(K, X, P, t) \mid \overline{t}(Y_k(K, X, P, t)))$$

with,

$$Y_k(K, X, P, t) \stackrel{def}{=} t(y) \triangleright (K[X] \triangleright (P \mid y \mid \overline{t}(y)))$$

For example, a process $stop(K) \diamond (K[x] \triangleright \mathbf{0})$ receives in channel $stop$ the name K of a kell; it matches the kell K's process to x, and reduces the kell to the null process $\mathbf{0}$. Such a process is useful to stop the execution of a kell:

$$\begin{array}{rl} & T[\overline{a}(b)] \mid \overline{stop}(T) \mid (stop(K) \diamond (K[x] \triangleright \mathbf{0})) \\ \rightarrow & T[\overline{a}(b)] \mid (stop(K) \diamond (K[x] \triangleright \mathbf{0})) \mid (T[x] \triangleright \mathbf{0}) \\ \rightarrow & (stop(K) \diamond (K[x] \triangleright \mathbf{0})) \mid \mathbf{0} \end{array}$$

In the previous example, the kell $T[\overline{a}(b)]$ is terminated by the process $\overline{stop}(T)$.

The symbol $\mid$ has lower precedence than, both, $\diamond$ and $\triangleright$. **new** has lower precedence than $\diamond$ and $\triangleright$, but higher than $\mid$. Associativity of $\mid$, $\diamond$, and $\triangleright$ is left-to-right. For example,

$$\textbf{new}\ e\ a(c) \triangleright c(d) \triangleright P \mid \overline{a}(d)$$

is equivalent to:

$$(\textbf{new}\ e\ (a(c) \triangleright (c(d) \triangleright P))) \mid \overline{a}(d)$$

3 Operational Semantics

Names in process expressions can be *bound* or *free*. Recall channels and kells are names. A free name is visible to any process. A bound name is visible only to the process expression where it is bound. The functions *fn* and *bn*, defined in Figure 2, produce the sets of free and bound names for kell-m processes.

Communication can happen between any two processes independently of their kell location. If a bound name is output via a channel, the name becomes visible to the receiving process. The term *scope extrusion* is used to specify this situation. For example, in the process expression:

$$(\textbf{new}\ c\ \overline{a}(c)) \mid a(d) \triangleright P$$

there is scope extrusion because the bound name c is output via channel a to a process $a(d) \triangleright P$, where c is not bound before the process receives c.

$$
\begin{array}{ll}
bn(\mathbf{0}) = \emptyset & fn(\mathbf{0}) = \emptyset \\
bn(x) = \emptyset & fn(x) = \{x\} \\
bn(\mathbf{new}\ a\ P) = \{a\} \cup bn(P) & fn(\mathbf{new}\ a\ P) = fn(P) \setminus \{a\} \\
bn(\overline{a}(\widetilde{w})) = bn(\widetilde{w}) & fn(\overline{a}(\widetilde{w})) = \{a\} \cup fn(\widetilde{w}) \\
bn(K[P]) = bn(P) & fn(K[P]) = \{K\} \cup fn(P) \\
bn(\widetilde{w}) = \bigcup_{w_i \in \widetilde{w}} bn(w_i) & fn(\widetilde{w}) = \bigcup_{w_i \in \widetilde{w}} fn(w_i) \\
bn(a(\widetilde{c}) \triangleright P) = \{\widetilde{c}\} \cup bn(P) & fn(a(\widetilde{c}) \triangleright P) = fn(P) \setminus \{\widetilde{c}\} \\
bn(K[x] \triangleright P) = \{x\} \cup bn(P) & fn(K[x] \triangleright P) = fn(P) \setminus \{x\} \\
bn(P \mid Q) = bn(P) \cap bn(P) & fn(P \mid Q) = fn(P) \cup fn(Q) \\
bn(p(\widetilde{w})) = bn(P_d\{\widetilde{w}/\widetilde{c}\}) & fn(p(\widetilde{w})) = fn(P_d\{\widetilde{w}/\widetilde{c}\}),\ \text{with}\ p(\widetilde{c}) \stackrel{def}{=} P_d
\end{array}
$$

Figure 2: Bound and Free Names in kell-m Processes

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

$$P \equiv P\{\widetilde{c}/\widetilde{d}\}, \text{with } \widetilde{d} \in bn(P) \text{ and } \widetilde{c} \notin fn(P) \qquad \mathbf{new}\ a,\ b\ P \equiv \mathbf{new}\ b,\ a\ P$$

$$\mathbf{new}\ a\ \ \mathbf{0} \equiv \mathbf{0} \qquad \frac{a \notin fn(Q)}{(\mathbf{new}\ a\ P) \mid Q \equiv \mathbf{new}\ a\ (P \mid Q)}$$

Figure 3: Structural Equivalences for kell-m

When a name c is bound in a process P, all occurrences of c can be replaced by b, written $P\{b/c\}$, if $b \notin fn(P)$. In terms of classical process theory, c is *alpha converted* to b [7]. The behaviour of P is not affected by the alpha conversion. In Section 3.3, we formally define what it means for P and $P\{b/c\}$ to have the same behaviour but, for now, the idea is that a process Q cannot differentiate whether it is interacting with P or $P\{b/c\}$. An interaction can be either a channel communication or a kell passivation.

Structural equivalences determine the processes for which their behavioural equivalency follows immediately from their structure [11]. We write $R \equiv S$ when processes R and S are structurally equivalent. Alpha conversion is an example of a structural equivalence. The structural equivalences for the kell-m calculus are specified in Figure 3: parallel composition of processes is commutative and associative, and the null process is its neutral element; restriction of a name in a null process is equivalent to the null process; and the scope of name restriction can be extended to the parallel composition if the restricted name is not free in the composed process.

3.1 LTS Semantics

We use a labelled transition system (LTS) to give the operational semantics of kell-m. The LTS describes the possible evolution of a process. Actions performed during the transitions can be: $\overline{a}(\widetilde{w})$, $a(\widetilde{c})$, $\overline{K}[P]$, $K[x]$, and τ:

- $\overline{a}(\widetilde{w})$ represents an output action on channel a.
- $a(\widetilde{c})$ represents an input action, via a matching trigger $a(\widetilde{c}) \triangleright R$, on channel a.
- $\overline{K}[P]$ represents an active kell with name K and process P.
- $K[x]$ represents the input of the process of kell K, via a matching trigger $K[x] \triangleright Q$.
- τ represents the matching of input and output actions on the same channel, or a kell passivation. A kell passivation corresponds to actions $\overline{K}[P]$ and $K[x]$ matching for the same kell K.

The transitions for kell-m are specified in Figure 4. The function *names* in the figure produces, for a process P, the set of free and bound names in P: $names(P) = bn(P) \cup fn(P)$.

Rule STRUCT, specifies that, if a process P transitions with action α to process R, any process Q, structurally equivalent to P, can also transition to R with action α.

$$\frac{P \xrightarrow{\alpha} R,\ P \equiv Q}{Q \xrightarrow{\alpha} R}\ \textsc{Struct}$$

$$\overline{a}(\widetilde{w}) \xrightarrow{\overline{a}(\widetilde{w})} \mathbf{0}\quad \textsc{Out} \qquad a(\widetilde{c}) \triangleright P \xrightarrow{a(\widetilde{c})} P\quad \textsc{In}$$

$$K[P] \xrightarrow{\overline{K}[P]} \mathbf{0}\quad \textsc{KellOut} \qquad K[x] \triangleright P \xrightarrow{K[x]} P\quad \textsc{KellIn}$$

$$\frac{P \xrightarrow{\alpha} Q,\ c \notin bn(\alpha)}{\mathbf{new}\ c\ P \xrightarrow{\alpha} \mathbf{new}\ c\ Q}\ \textsc{Restrict} \qquad \frac{P \xrightarrow{\alpha} Q,\ K \notin bn(\alpha)}{K[P] \xrightarrow{\alpha} K[Q]}\ \textsc{Advance}$$

$$\frac{P \xrightarrow{\alpha} Q, bn(\alpha) \cap fn(R) = \emptyset}{P|R \xrightarrow{\alpha} Q|R}\ \textsc{Par} \qquad \frac{P_d\{\widetilde{w}/\widetilde{x}\} \xrightarrow{\alpha} Q,\ p(\widetilde{x}) \stackrel{def}{=} P_d}{p(\widetilde{w}) \xrightarrow{\alpha} Q}\ \textsc{Proc}$$

$$\frac{P \xrightarrow{\overline{a}(\widetilde{w})} Q,\ c \in names(\widetilde{w}),\ c \neq a}{\mathbf{new}\ c\ P \xrightarrow{\overline{a}(\widetilde{w}')} Q,\ \text{with } \widetilde{w}' = \widetilde{w}\{\mathbf{new}\ c\ /c\}}\ \textsc{Open}$$

$$\frac{P \xrightarrow{a(\widetilde{c})} P',\ Q \xrightarrow{\overline{a}(\widetilde{w})} Q'}{P \mid Q \xrightarrow{\tau} P'\{\widetilde{w}/\widetilde{c}\} \mid Q'}\ \textsc{L-React} \qquad \frac{P \xrightarrow{K[x]} P',\ Q \xrightarrow{\overline{K}[R]} Q'}{P \mid Q \xrightarrow{\tau} P'\{R/x\} \mid Q'}\ \textsc{L-Suspend}$$

$$\frac{P \xrightarrow{a(\widetilde{d})} P',\ Q \xrightarrow{\overline{a}(\widetilde{w})} Q',\ \widetilde{c} \subseteq \widetilde{w},\ (\mathbf{new}\ c\ ) \in \widetilde{w} \text{ if } c \in \widetilde{c}}{P \mid Q \xrightarrow{\tau} \mathbf{new}\ \widetilde{c}\ (P'\{\widetilde{w}/\widetilde{c}\} \mid Q')}\ \textsc{L-Close}$$

Figure 4: Labelled Transition System Semantics for kell-m

Rules OUT, IN, KELLOUT, and KELLIN, specify transitions due to basic communication actions. In terms of classical process theory [7], both $\overline{a}(\widetilde{w})$ and $K[P]$ correspond to *concretions*. Trigger matching expressions $a(\widetilde{c})$ and $K[x]$ in triggers $a(\widetilde{c}) \triangleright P$, and $K[x] \triangleright P$ correspond to *abstractions*.

Rule RESTRICT specifies that a transition α occurs in P and **new** c P, if c, the name restricted, is not bound in the transition action. Since *bn* was defined for process expressions (cf. Figure 2), not for transition actions, here we are abusing its definition. Hence, we extend *bn* to deal with transition actions as follows:

$$\begin{aligned} bn(a(\widetilde{c})) &= \{\widetilde{c}\} \\ bn(K[x]) &= \{x\} \\ bn(\overline{a}(\widetilde{w})) &= bn(\widetilde{w}) \\ bn(\overline{K}[P]) &= bn(P) \end{aligned}$$

Rule ADVANCE specifies that if a process P can transition to a process Q, the same process P, when within a kell K, can also transition with the same action α. This is the case when a kell is not passivated: the process located within the kell can transition (*advance*). Once the process has advanced, it can, again, be passivated or advance.

Although not obvious at this point of the presentation, the rules KELLOUT and ADVANCE provide an intuitive notion for the semantics of kells in kell-m. These two rules specify that every active kell $K[P]$ can be seen as a higher-order processes that, non deterministically, either outputs P on a channel named K, or advances in the evolution of P. Later in this report (cf. Section 5), we make use of this intuitive notion to encode kell-m using a variation of the π-calculus. We also show that this intuition is valid, in the sense that an observer is not able to distinguish between a kell-m process and its π-calculus encoding.

When the bound names in action α do not occur free in process R, and a process P transitions with action α to process Q, the rule PAR specifies that the process resulting from the parallel composition of P and another process R, can transition with the same action α to process $Q|R$. The condition $bn(\alpha) \cap fn(R) = \emptyset$ guarantees that free names in R are not, inadvertently, captured in an alpha conversion after a communication, as illustrated by the following sample processes:

$$P : a(c) \triangleright \mathbf{0} \qquad R : \overline{b}(c) \qquad T : \overline{a}(e)$$

Because of rule IN, $P \xrightarrow{a(c)} \mathbf{0}$, and because of rule OUT, $T \xrightarrow{\overline{a}(e)} \mathbf{0}$. If $bn(\alpha) \cap fn(R) = \emptyset$ is not required in PAR, then $P|R \xrightarrow{a(c)} \mathbf{0}|R$. Because of rule L-REACT:

$$\frac{P|R \xrightarrow{a(c)} \mathbf{0}, T \xrightarrow{\overline{a}(e)} \mathbf{0}}{(P|R) \mid T \xrightarrow{\tau} (\mathbf{0}|R)\{e/c\} \mid \mathbf{0}} \text{ L-REACT}$$

But, since $c \in fn(R)$, c is captured by the alpha conversion $(\mathbf{0}|R)\{e/c\}$, resulting in $(\mathbf{0}|\overline{b}(e))$, which is incorrect.

Processes can be parameterized using the syntax $p(\widetilde{x}) \stackrel{def}{=} P_d$. Rule PROC, deals with these definitions. The rule specifies that a process invocation $p(\widetilde{w})$ can transition to Q, if the process definition P_d, when the parameters $\widetilde{x}$ have been replaced by the values $\widetilde{w}$, can also transition to Q.

The rule OPEN deals with scope extrusion. If a restricted name c is output on an channel, the list of output values $\widetilde{w}$ is modified by replacing c with **new** c. The rule L-CLOSE specifies what happens when the restricted name is received: the name must continue to be restricted and its scope now reaches the reader and writer processes.

Rules L-REACT and L-SUSPEND specify the cases when a communication occurs on a channel, and when a passivation occurs. R-* transition rules can be trivially deduced by first using the STRUCT rule, and then the corresponding L-* rule. When illustrating transitions for process expressions, we sometimes write the name of the channel involved in a communication action right after the τ, e.g., $P \xrightarrow{\tau,a} Q$.

We refer to rules *-REACT, *-SUSPEND and *-CLOSE as the *communication rules*. These are the rules where abstractions and concretions are matched. The transitions in these rules are decorated with τ.

To illustrate the use of the transition rules, consider the process P defined as:

$$P : stop(K) \triangleright (K[x] \triangleright \mathbf{0}) \mid T[t(x) \triangleright x \mid \overline{stop}(F)] \mid F[f(x) \triangleright x \mid \overline{stop}(T)]$$

Any process received on channel t or f is executed. If channel t is used, channel f is discarded, and vice versa. Such a process is useful to represent conditionals. In Figure 5, we show that $\mathbf{0} \mid \mathbf{0} \mid T[P_t|\mathbf{0}] \mid \mathbf{0}$ can be inferred from $\overline{t}(P_t) \mid P$. In Figure 5 we have rearranged the process expressions to facilitate the drawing of an inference tree; process expressions involved in the transitions are double-underlined.

We use the notation $K^n[P]$ to specify a kell K and its process P when the kell is embedded within $n-1$ other kells:

$$\begin{array}{ll} K^1[P] & \text{when } K[P] \\ K^2[P] & \text{when } \exists K_1 : K_1[K[P] \cdots] \\ \cdots & \\ K^n[P] & \text{when } \exists K_1, ..., k_{n-1} : K_1[K_2[\cdots K_{n-1}[K[P] \cdots] \cdots] \cdots] \end{array}$$

We call n the *depth-level* of kell K, and write $K^*[P]$ to specify a kell-m process at any depth-level. In particular, we write $K^0[P]$ when P is not within a kell.

For notational convenience, we introduce generalized versions of the communication rules *-REACT, *-SUSPEND and *-CLOSE as specified in Figure 6 (only the left-hand versions are shown, right-hand versions are similarly defined). The generalized rules are equivalent to applications of the PAR and ADVANCE rules, before using the communication rules.

3.2 Reduction Semantics

Reduction semantics are an alternative to LTS semantics. In reduction semantics, reduction rules are used to describe the operational semantics of kell-m. The reduction rules for kell-m are listed in Figure 7. With reduction rules the communication between processes is inferred, directly, from the syntax of the processes instead of from transitions where abstraction and concretion actions occur.

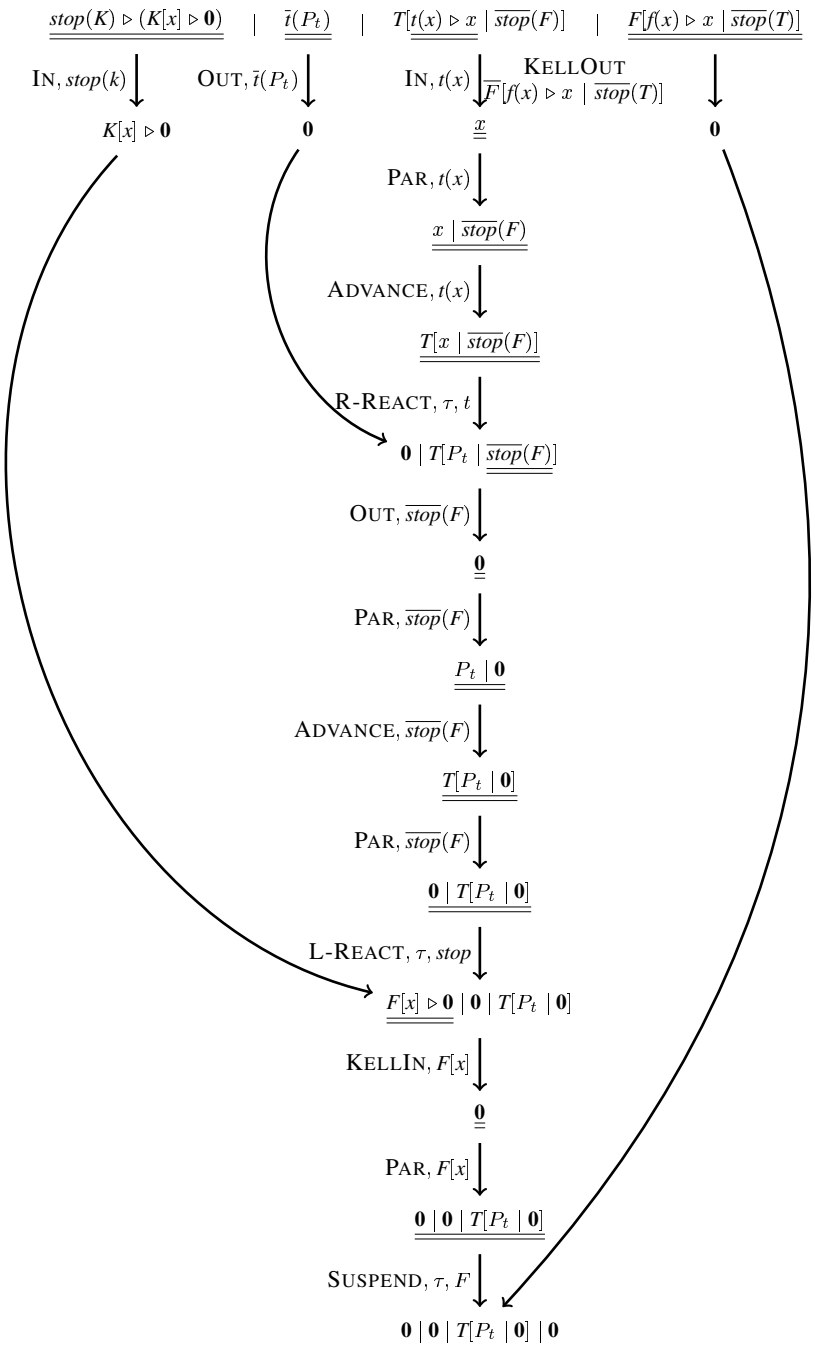

Figure 5: Sample Process Evolution Using LTS Semantics

$$\frac{P \xrightarrow{a(\widetilde{c})} P',\ Q \xrightarrow{\overline{a}(\widetilde{w})} Q'}{K^*[P] \mid M^*[Q] \xrightarrow{\tau} K^*[P'\{\widetilde{w}/\widetilde{c}\}] \mid M^*[Q']}\ \text{L-REACT}$$

$$\frac{P \xrightarrow{K[x]} P',\ Q \xrightarrow{\overline{K}[R]} Q'}{K^*[P] \mid M^*[Q] \xrightarrow{\tau} K^*[P'\{R/x\}] \mid M^*[Q']}\ \text{L-SUSPEND}$$

$$\frac{P \xrightarrow{a(\widetilde{d})} P',\ Q \xrightarrow{\overline{a}(\widetilde{w})} Q',\ \widetilde{c} \subseteq \widetilde{w},\ (\textbf{new}\ c\ ) \in \widetilde{w} \text{ if } c \in \widetilde{c}}{K^*[P] \mid M^*[Q] \xrightarrow{\tau} \textbf{new}\ \widetilde{c}\ (K^*[P'\{\widetilde{w}/\widetilde{c}\}] \mid M^*[Q'])}\ \text{L-CLOSE}$$

Figure 6: Generalized Communication Rules for kell-m

$$\frac{}{a(\widetilde{c}) \triangleright P \mid \overline{a}(\widetilde{w}) \rightarrowtail P\{\widetilde{w}/\widetilde{c}\} \mid \mathbf{0}}\ \text{L-REDUCTREACT} \qquad \frac{}{M[x] \triangleright P \mid M[P_M] \rightarrowtail P\{P_M/x\} \mid \mathbf{0}}\ \text{L-REDUCTSUSPEND}$$

$$\frac{P \rightarrowtail P'}{\textbf{new}\ c\ P \rightarrowtail \textbf{new}\ c\ P'}\ \text{REDUCTRESTRICT} \qquad \frac{P \rightarrowtail P'}{K[P] \rightarrowtail K[P']}\ \text{REDUCTINKELL} \qquad \frac{P \rightarrowtail P'}{P|Q \rightarrowtail P'|Q}\ \text{REDUCTPAR}$$

$$\frac{P' \equiv P,\ P \rightarrowtail Q,\ Q \equiv Q'}{P' \rightarrowtail Q'}\ \text{REDUCTSTRUCT} \qquad \frac{P_d\{\widetilde{w}/\widetilde{c}\} \rightarrowtail Q,\ p(\widetilde{c}) \stackrel{def}{=} P_d}{p(\widetilde{w}) \rightarrowtail Q}\ \text{REDUCTPROC}$$

$$\frac{P \mid Q \rightarrowtail P' \mid Q'}{K^*[P \cdots] \mid M^*[Q \cdots] \rightarrowtail K^*[P' \cdots] \mid M^*[Q' \cdots]}\ \text{REDUCTOUTKELL}$$

$$\frac{(\textbf{new}\ \widetilde{c}\ P) \mid Q \rightarrowtail \textbf{new}\ \widetilde{c}\ (P' \mid Q')}{K^*[\textbf{new}\ \widetilde{c}\ P \cdots] \mid M^*[Q \cdots] \rightarrowtail \textbf{new}\ \widetilde{c}\ (K^*[P' \cdots] \mid M^*[Q' \cdots])}\ \text{L-REDUCTEXTRUSION}$$

Figure 7: Reduction Rules for kell-m

As shown in rule REDUCTOUTKELL, reactions and suspensions can occur within kells. In this rule, $\cdots$ are used to represent zero or more (bound name set, kell-m process)-pairs composed in parallel.

In the reduction rules, we assume there are no name conflicts for bound names. As we discuss at the beginning of Section 3 when presenting the structural equivalences for kell-m, since $K[\textbf{new}\ a\ P] \not\equiv \textbf{new}\ a\ (K[P])$, name restrictions within kells need to be treated with care. The reduction rule L-REDUCTEXTRUSION explicitly handles name extrusion from kells.

When illustrating reductions of process expressions, we sometimes write the name of the channel involved in a communication action, e.g., $P \rightarrowtail {}^{a}Q$.

R-$*$ reduction rules can be trivially deduced by first using the REDUCTSTRUCT rule, and then the corresponding L-$*$ rule.

For example, a process P defined as:

$$P : K[\textbf{new}\ a, b\ (\overline{c}(\overline{a}(b)) \mid a(d) \triangleright Q)] \mid c(x) \triangleright x$$

can be reduced as follows:

$$\begin{array}{rll} P \rightarrowtail^{c} & \textbf{new}\ a, b\ (K[\mathbf{0} \mid a(d) \triangleright Q] \mid \overline{a}(b)) & \text{L-REDUCTEXTRUSION} \\ \rightarrowtail^{a} & \textbf{new}\ a, b\ (K[\mathbf{0} \mid Q\{b/d\}] \mid \mathbf{0}) & \text{REDUCTOUTKELL} \end{array}$$

$$
\begin{array}{lcl}
\mathcal{V}[\![\ \mathbf{0}\ ]\!] & \stackrel{def}{=} & \{\} \\
\mathcal{V}[\![\ x\ ]\!] & \stackrel{def}{=} & \{\} \\
\mathcal{V}[\![\ \mathbf{new}\ a\ P\ ]\!] & \stackrel{def}{=} & \mathcal{V}[\![\ P\ ]\!] \setminus \{a\} \\
\mathcal{V}[\![\ \overline{a}(\widetilde{w})\ ]\!] & \stackrel{def}{=} & \{\overline{a}\} \\
\mathcal{V}[\![\ K[P]\ ]\!] & \stackrel{def}{=} & \{\overline{K}\} \cup \mathcal{V}[\![\ P\ ]\!] \\
\mathcal{V}[\![\ a(\widetilde{c}) \triangleright P\ ]\!] & \stackrel{def}{=} & \{a\} \\
\mathcal{V}[\![\ K[x] \triangleright P\ ]\!] & \stackrel{def}{=} & \{K\} \\
\mathcal{V}[\![\ P|Q\ ]\!] & \stackrel{def}{=} & \mathcal{V}[\![\ P\ ]\!] \cup \mathcal{V}[\![\ Q\ ]\!]
\end{array}
$$

Figure 8: Visibility Predicates for kell-m

When dealing with higher-order expressions, the scope extrusion occurs for any name in the expression that is bound when the expression is output via a channel ($\overline{c}(\overline{a}(b))$ in the example).

As observed in [11], although in process algebras reduction rules are simpler than LTS semantics in the sense that there are typically less reduction rules than LTS transition rules and the reduction rules are unlabelled, there is loss of information when compared to LTS transitions. For example, consider the process $\overline{a}(w)$. This process cannot be reduced. It, nevertheless, has the potential to communicate with another process via the channel a. Such potential for communication is manifest in the OUT LTS transition, but is lost when using reduction semantics. As we will see in Section 3.3, this lost information is somehow recovered when assuming a global observer capable of sensing, at a given time, which channels are capable of communication.

3.3 Behavioural Equivalences

We are interested in knowing if two kell-m processes exhibit the same behaviour. As proposed by Sangiorgi when dealing with higher-order calculi [12], we assume a global observer capable of sensing the communication actions that are enabled on a given channel. One can think of the global observer as another process trying to interact with the observed process.

We write $P \downarrow_{\overline{a}}$ (similarly, $P \downarrow_a$) to specify that a concretion (similarly, abstraction) is enabled on channel a. $P \downarrow_{\overline{a}}$ and $P \downarrow_a$ are called *visibility predicates*, and we frequently write $P \downarrow_\alpha$ to indicate an arbitrary visibility predicate.

The visibility predicates for kell-m are determined by the function $\mathcal{V}$, defined in Figure 8. According to $\mathcal{V}$'s definition, $P \downarrow_\alpha$, if $\alpha \in \mathcal{V}[\![\ P\ ]\!]$. Hence, $a \in \mathcal{V}[\![\ P\ ]\!]$ if exists P', such that $P \xrightarrow{a(\widetilde{c})} P'$ or $P \xrightarrow{a[x]} P'$. Both, $a(\widetilde{c})$ and $a[x]$, are abstractions representing patterns in trigger expressions. $a(\widetilde{c})$ matches a write on channel a, and $a[x]$ matches a kell named a. Similarly, $\overline{a} \in \mathcal{V}[\![\ P\ ]\!]$ if $P \xrightarrow{\overline{a}(\widetilde{w})} P'$ or $P \xrightarrow{\overline{a}[Q]} P'$. Both, $\overline{a}(\widetilde{w})$ and $\overline{a}[Q]$ are concretions. $\overline{a}(\widetilde{w})$ represents a write on channel a, and $\overline{a}[Q]$ represents a kell $a[Q]$.

Notice restricted names (names in **new** ... expressions) are not visible. Also notice that τ is not included as a visibility predicate. The reason is that τ actions in the observed process are actions internal to the process and cannot be sensed by the global observer.

3.3.1 Barbed Bisimulation

Using P^k to represent the class of kell-m processes, and based on Sangiorgi's behavioural equivalences for higher-order process calculi [12], a *barbed simulation* for the kell-m calculus is any relation S such that $S \subseteq P^k \times P^k$, and $(P, Q) \in S$ if:

1. $P \xrightarrow{\tau} P'$, then $\exists Q' : Q \xrightarrow{\tau} Q'$, and $(P', Q') \in S$, and
2. If $P \downarrow_\alpha$, then $Q \downarrow_\alpha$

Alternatively, using the reduction rules instead of the LTS semantics, condition 1 in the definition of barbed simulation relations can be replaced by:

1. $P \rightarrowtail P'$, then $\exists Q' : Q \rightarrowtail Q'$ $(P', Q') \in S$, and

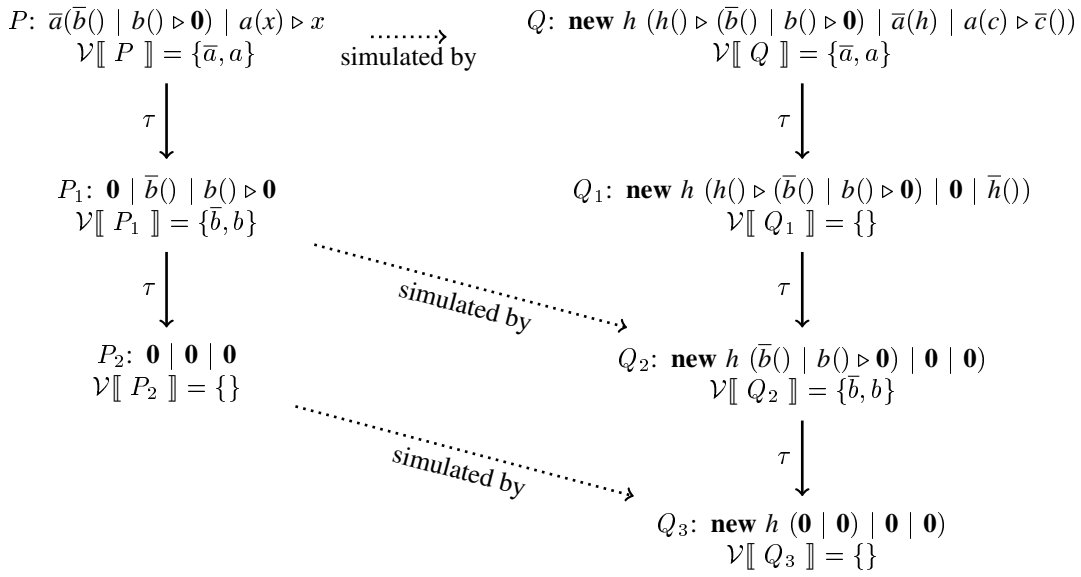

Figure 9: Example of Weak Barbed Simulation in kell-m

condition 2 remains unaltered.

The reason why barbed simulation can be defined with the communication transitions $\xrightarrow{\tau}$ or using the reduction relation, is that the reduction relations represent, the $\xrightarrow{\tau}$ transitions of the labelled transition semantics, except for the differences accounted by structural equivalences [9].

A relation S is a *barbed bisimulation*, if both S and S^{-1} are barbed simulations. *Weak* definitions of the barbed simulation and bisimulations are obtained by replacing $\rightarrow$ with $\Rightarrow$ in the definitions, where $\Rightarrow$ is the reflexive and transitive closure of $\rightarrow$ (or $\rightarrowtail$ if using the reduction relation).

A process P is said to be *barbed similar* to a process Q if there is a barbed simulation S such that $(P, Q) \in S$. Similarly, P and Q are *barbed bisimilar* if there is a barbed bisimulation S such that $(P, Q) \in S$.

For example, consider processes P and Q defined as:

$$\begin{array}{ll} P: & \overline{a}(\overline{b}() \mid b() \triangleright \mathbf{0}) \mid a(x) \triangleright x \\ Q: & \textbf{new}\ h\ (h() \triangleright (\overline{b}() \mid b() \triangleright \mathbf{0}) \mid \overline{a}(h) \mid a(c) \triangleright \overline{c}()) \end{array}$$

Intuitively, Q and P have the same behaviour: the potential for communication on a is visible in both process, and later, the potential for communication on b is visible. The communication transitions and visibility predicates for each one of the processes are illustrated in Figure 9.

Notice in Figure 9 Q is not barbed similar to P. The reason is the transition from Q_1 to Q_2 which is not matched in P. A weak barbed simulation S for P and Q is $\{(P, Q), (P_1, Q_2), (P_2, Q_3)\}$. S is represented in the figure by the dotted arrows.

Even when two processes are barbed bisimilar, their behaviour may not be the same when used along other processes. For example, consider the following processes:

$$\begin{array}{ll} P: & \overline{a}(c) \mid \overline{b}(u) \mid b(l) \triangleright \mathbf{0} \\ Q: & \overline{a}(d) \mid \overline{b}(u) \mid b(w) \triangleright \mathbf{0} \end{array}$$

A barbed bisimulation for them is:

$$S = \{(P, Q), (\overline{a}(c) \mid \mathbf{0} \mid \mathbf{0}, \overline{a}(d) \mid \mathbf{0} \mid \mathbf{0})\}$$

But when used along the process $R : a(e) \triangleright \overline{e}()$, the visibility predicates of $P|R$ and $Q|R$ are different. After P and R interact, the visibility predicate is $\{\overline{c}, \overline{b}, b\}$; if Q and R interact, the visibility predicate is $\{\overline{d}, \overline{b}, b\}$.

To define behavioural similarity under any use, it is necessary to introduce contexts $\mathcal{C}$. Assuming arbitrary kell-m processes R and P, P can be used in the following contexts $\mathcal{C}$:

$$\mathcal{C} \quad ::= \quad P \mid \textbf{new}\ a\ P \mid \overline{a}(P) \mid P|R \mid R|P \mid a(\widetilde{c}) \triangleright \mathcal{C} \mid K[\mathcal{C}] \mid K[x] \triangleright \mathcal{C}$$

Two processes P and Q are *barbed congruent* if they are barbed bisimilar under all contexts. When two processes are barbed congruent, they can be used interchangeably. Showing that two processes are barbed congruent requires showing that there is no context under which the processes behave differently. A differentiating context may be easy to find for some processes that are not congruent, but proving that such a context does not exist for processes suspected congruent can be a difficult task [14]. Although we were unable to find in the literature an actual result for the complexity of this task, Davide Sangiorgi has this to say [14]:

> Context-based behavioural equalities like barbed congruence suffer from the universal quantification on contexts, that makes it very hard to prove process equalities following the definition, and makes mechanical checking impossible.

We conclude the presentation of barbed bisimulation in kell-m by demonstrating that barbed congruent processes have the same kell structure. Recall the kell structure of a process determines the containment relationships between kells in the process. By contradiction, let us assume processes P and Q are barbed congruent but they do not have the same kell structure. If P and Q do not have the same kell structure, it is because one of the following cases:

- There is at least a kell K in one processes and not in the other. Without loss of generality, let us assume K is in P and not in Q. A differentiating context is $C_1 : K[x] \triangleright \mathbf{0}$. This is because $P|C$ has at least one transition labelled τ due to the passivation of K. That transition does not occur in $Q|C$. Moreover, K is in the visibility predicate for $P|C$ but not for $Q|C$.
- A kell K is located in another kell L in one of the processes but not in the other. Both processes P and Q have kells K and L, otherwise the previous case applies. Without loss of generality, let us assume kell K is located in kell L in P but not in Q. A differentiating context in this case is $C_2 : L[x] \triangleright K[y] \triangleright \mathbf{0}$. With this context, $Q|C$ has an extra transition labelled τ, corresponding to the passivation of K, that does not occur in $P|C$. The reason is that in $P|Q$, because K is located in L, the passivation of K cannot occur once L is passivated.

In both cases we were able to exhibit differentiating contexts. Hence, if P and Q are barbed congruent, then they have the same kell structure.

Besides C_1 and C_2 above, other differentiating contexts may exist for particular processes P and Q. For example, consider:

$$P : K[a(c) \triangleright \mathbf{0}] \mid T[\mathbf{0}] \mid \overline{a}(w)$$
$$Q : K[T[a(c) \triangleright \mathbf{0}]] \mid \overline{a}(w)$$

Processes P and Q are barbed bisimilar: a barbed bisimulation for them is:

$$S = \{(P, Q), (K[\mathbf{0}] \mid T[\mathbf{0}] \mid \mathbf{0}, K[T[\mathbf{0}]] \mid \mathbf{0})\}$$

A differentiating context is $T[x] \triangleright \mathbf{0}$. $P|C$ produces $K[a(c) \triangleright 0] \mid \mathbf{0} \mid \overline{a}(w)$, while $Q|C$ produces $K[\mathbf{0}] \mid \overline{a}(w)$. These resulting processes are not barbed bisimilar.

3.3.2 Bisimulation up to Kell Containment

We are interested in verifying if two processes have the same kell structure. As discussed at the end of the previous section, an avenue is to check if they are barbed congruent. Although, because of its difficulty, relying on barbed congruence should be avoided. Moreover, barbed congruency is a sufficient, but not a required, condition for kell structure similarity. For example, consider:

$$Q : K[T[a(c) \triangleright \mathbf{0}]] \mid \overline{a}(w)$$

Processes $Q|\overline{b}(e)$ and $Q|\overline{b}(l)$ are not barbed congruent. A differentiating context in this case is C: $b(r) \triangleright \overline{r}()$. They have, nevertheless, the same kell structure.

Using P^k to represent the class of kell-m processes, a *simulation up to kell containment* for kell-m is any relation S such that, $S \subseteq P^k \times P^k$, and $(P, Q) \in S$ if:

$$P \xrightarrow{\overline{K}[.]} P', \text{ then } \exists Q' : Q \xrightarrow{\overline{K}[.]} Q', \text{ and } (P', Q') \in S$$

The dot is used to indicate that we are not interested in the contents of the concretion.

A relation S is a *bisimulation up to kell containment*, if both S and S^{-1} are barbed simulations up to kell containment. As with the regular barbed bisimulations, *weak* definitions of the barbed simulation and bisimulations up to kell containment are obtained by replacing $\rightarrow$ with $\Rightarrow$ in the definitions, where $\Rightarrow$ is the reflexive and transitive closure of $\rightarrow$.

Two processes have the same structure if and only if they are bisimilar up to kell containment. We first show that if processes P and Q are bisimilar up to kell containment, then they have the same kell structure. We argue by contradiction. Let us assume that P and Q are bisimilar up to kell containment but they do not have the same kell structure. The possible situations under which P and Q do not have the same kell structure are:

- There is a kell K in one of the processes but not in the other. Without loss of generality, let us assume K is in P but not in Q. Because of rule KELLOUT (cf. Figure 4), a possible transition for P is $P \xrightarrow{\overline{K}[R]} P'$, where R is the process located in K. Since K is not in Q such transition does not exist in Q and, by definition, P and Q cannot be bisimilar up to kell containment.

- A kell K is located within a kell T in one of the processes but not in the other. Without loss of generality we assume kell K is located in kell T in P but not in Q. Both kells, K and T, are in Q, otherwise the previous case applies. Because of rule KELLOUT, P can transition to $P \xrightarrow{\overline{T}[U]} P'$ where R is the process located in T. By the definition of bisimilarity up to kell containment, Q can also transition to Q': $Q \xrightarrow{\overline{T}[U]} Q'$. Since T has been passivated, and because of our assumption about the location of K in P, K is not in P'. Because our assumption of the location of K in Q, K has not been passivated in Q', and Q' can transition to $Q' \xrightarrow{\overline{K}[U']} Q''$, which is a transition that cannot be matched from P'. Therefore, by definition, P and Q cannot be bisimilar up to kell containment.

We now show that if processes P and Q have the same kell structure, then they are bisimilar up to kell containment. Again, by contradiction, let us assume that P and Q have the same kell structure but they are not bisimilar up to kell containment. If P and Q are not bisimilar up to kell containment this is because one of the following reasons:

- There is a transition from $P \xrightarrow{\overline{K}[R]} P'$ that is not matched from Q. This means a kell K is in P and not in Q. Therefore P and Q do not have the same kell structure.

- There is a transition from $Q \xrightarrow{\overline{K}[R]} Q'$ that is not matched from P. The argument is the same as the previous case.

- P and Q have matching transitions up to processes P'' and Q'' at which point there is a transition from P'' that cannot be matched from Q''. That transition represents a kell that is in P and not in Q. Therefore P and Q do not have the same kell structure.

- P and Q have matching transitions up to processes P'' and Q'' at which point there is a transition from Q'' that cannot be matched from P''. The argument is the same as the previous case.

Hence, to verify if two processes have the same kell structure, a bisimulation up to kell containment has to be produced.

3.4 Extended Semantics

It is useful not only to check if two processes have the same kell structure, but also to have the ability to verify kell containment conditions. For example, one may be interested in checking if a given action occurs, or does not occur, within a given kell. Based on the operational semantics for kell-m we have presented so far, there is no simple way to check such kell containment conditions.

$$\frac{P \xrightarrow{\delta} R,\ P \equiv Q}{Q \xrightarrow{\delta} R}\ \textsc{XStruct} \qquad \overline{a}(\widetilde{w}) \xrightarrow{\overline{a}(\widetilde{w}),\{\}} \mathbf{0}\ \textsc{XOut} \qquad a(\widetilde{c}) \triangleright P \xrightarrow{a(\widetilde{c}),\{\}} P\ \textsc{XIn}$$

$$K[P] \xrightarrow{\overline{K}[P],\{\}} \mathbf{0}\ \textsc{XKellOut} \qquad K[x] \triangleright P \xrightarrow{K[x],\{\}} P\ \textsc{XKellIn} \qquad \frac{P \xrightarrow{\alpha,\kappa} Q,\ c \notin bn(\alpha)}{\mathbf{new}\ c\ P \xrightarrow{\alpha,\kappa\setminus\{c\}} \mathbf{new}\ c\ Q}\ \textsc{XRestrict}$$

$$\frac{P \xrightarrow{\alpha_\tau,\kappa_a,\kappa_c} Q}{\mathbf{new}\ c\ P \xrightarrow{\alpha_\tau,\kappa_a\setminus\{c\},\kappa_c\setminus\{c\})} \mathbf{new}\ c\ Q}\ \textsc{XRestrictTau} \qquad \frac{P \xrightarrow{\alpha,\kappa} Q,\ K \notin bn(\alpha)}{K[P] \xrightarrow{\alpha,\kappa\cup\{K\}} K[Q]}\ \textsc{XAdvance}$$

$$\frac{P \xrightarrow{\alpha_\tau,\kappa_a,\kappa_c} Q}{K[P] \xrightarrow{\alpha_\tau,\kappa_a\cup\{K\},\kappa_c\cup\{K\}} K[Q]}\ \textsc{XAdvanceTau} \qquad \frac{P \xrightarrow{\delta} Q, (\delta = (\alpha,\kappa)) \Rightarrow (bn(\alpha) \cap fn(R) = \emptyset)}{P|R \xrightarrow{\delta} Q|R}\ \textsc{XPar}$$

$$\frac{P_d\{\widetilde{w}/\widetilde{x}\} \xrightarrow{\delta} Q,\ p(\widetilde{x}) \stackrel{def}{=} P_d}{p(\widetilde{w}) \xrightarrow{\delta} Q}\ \textsc{XProc} \qquad \frac{P \xrightarrow{\overline{a}(\widetilde{w}),\kappa} Q,\ c \in names(\widetilde{w}),\ c \neq a}{\mathbf{new}\ c\ P \xrightarrow{\overline{a}(\widetilde{w}'),\kappa\setminus\{c\}} Q,\ \text{with } \widetilde{w}' = \widetilde{w}\{\mathbf{new}\ c\ /c\}}\ \textsc{XOpen}$$

$$\frac{P \xrightarrow{a(\widetilde{c}),\kappa_a} P',\ Q \xrightarrow{\overline{a}(\widetilde{w}),\kappa_c} Q'}{P \mid Q \xrightarrow{\overleftrightarrow{a}(\widetilde{w}),\kappa_a,\kappa_c} P'\{\widetilde{w}/\widetilde{c}\} \mid Q'}\ \text{XL-React} \qquad \frac{P \xrightarrow{K[x],\kappa_a} P',\ Q \xrightarrow{\overline{K}[R],\kappa_c} Q'}{P \mid Q \xrightarrow{\overleftrightarrow{K}[R],\kappa_a,\kappa_c)} P'\{R/x\} \mid Q'}\ \text{XL-Suspend}$$

$$\frac{P \xrightarrow{a(\widetilde{d}),\kappa_a} P',\ Q \xrightarrow{\overline{a}(\widetilde{w}),\kappa_c} Q',\ \widetilde{c} \subseteq \widetilde{w},\ (\mathbf{new}\ c\ ) \in \widetilde{w} \text{ if } c \in \widetilde{c}}{P \mid Q \xrightarrow{\overleftrightarrow{a}(\widetilde{w}),\kappa_a,\kappa_c} \mathbf{new}\ \widetilde{c}\ (P'\{\widetilde{w}/\widetilde{c}\} \mid Q')}\ \text{XL-Close}$$

Figure 10: Kell-m Extended Labelled Transition System Semantics

Recall the LTS semantics for kell-m, presented in Section 3.1 and listed in Figure 4. We now extend kell-m's LTS operational semantics with kell containment information, and expose the concretion and abstraction actions matched in τ communications. The rules for the extended LTS semantics are listed in Figure 10. Specifically, the changes are:

- For abstraction and concretion actions, a set κ is included with the names of kells where the action is located.
- τ transitions are decorated with the name of the channel or kell involved in the communication, the parameters of the communication, and the kell containment sets for the matched abstraction and concretion.
- To further differentiate τ actions from abstraction and concretions, channel and kell names in the communication are decorated with an over double arrow, e.g., $\overleftrightarrow{a}(\widetilde{w})$, $\overleftrightarrow{K}[P]$.
- α in the transitions represents abstractions (i.e., $a(\widetilde{c})$, $K[x]$), concretions (i.e., $\overline{a}(\widetilde{w})$, $K[P]$). α_τ represents τ actions (i.e., $\overleftrightarrow{a}(\widetilde{w})$, $\overleftrightarrow{K}[P]$)
- δ is used to represent any kind of transition. Therefore, δ depicts a triple $(\alpha_\tau, \kappa_a, \kappa_c)$ for τ actions, and a pair (α, κ) for abstractions and concretions.
- To differentiate the new rules from the rules in Section 3.1, we prefix the new rules with the letter X.

We will assume XR-*, similar to the communication rules XL-REACT, XL-SUSPEND, and XL-CLOSE, but with the order of the reader and writer processes inverted. The XR-* are obtained by applying the XSTRUCT rule, and then the corresponding XL-* rule.

XRESTRICT removes restricted names from the kell containment set. When a process is located within a kell, if the process can advance with concretion or abstraction actions, rule XADVANCE adds the kell to the kell containment set.

Using extended semantics, Figure 11 shows the possible evolution of a processes $Q : K[T[a(c) \triangleright \mathbf{0}]] \mid L[\overline{a}(w)]$.

The extended LTS semantics are the basis of kμ, our logic for the specification of properties in systems modelled using kell-m (kμ is presented in Section 4 below). When τ actions expose the information of the abstraction and

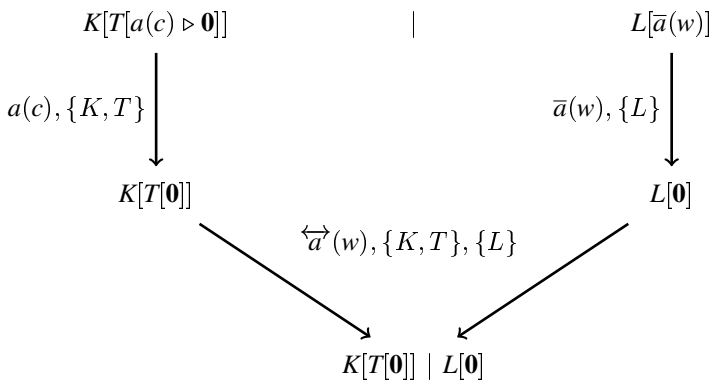

Figure 11: Example of Process Evolution with Extended LTS Semantics

$$
\begin{array}{lll}
\mathcal{F} & ::= & \texttt{tt} \mid \texttt{ff} \mid \neg\mathcal{F} \mid \mathcal{C}.\mathcal{F} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \langle\eta\rangle.\mathcal{F} \mid [\eta].\mathcal{F} \mid N(\widetilde{p}) \\
\mathcal{C} & ::= & a \; op \; b \mid a \in \widetilde{w} \mid |\mathcal{K}| = n \mid a \in \mathcal{K} \mid \neg\mathcal{C} \mid \mathcal{C} \wedge \mathcal{C} \mid \mathcal{C} \vee \mathcal{C} \\
\eta & ::= & \varphi \mid -\varphi \mid S \mid -S \\
\varphi & ::= & (\alpha, \gamma) \mid (\alpha_\tau, \gamma, \gamma) \\
\gamma & ::= & * \mid \mathcal{K} \mid \supseteq \mathcal{K} \mid \not\subseteq \mathcal{K} \mid I \\
N(\widetilde{p}) & ::= & \mathcal{F}\{\widetilde{p}/\widetilde{x}\} \text{ with } N(\widetilde{x}) \stackrel{def}{=} \mathcal{F} \\
op & ::= & = \mid \neq \mid > \mid < \mid \geq \mid \leq
\end{array}
$$

Figure 12: kμ Syntax

concretion actions being matched, they are no longer invisible to an observer. With kμ, this allows the specification of properties that can impose conditions at the time processes communicate. For example, one may be interested in specifying a condition such as *process in kell K_1 communicates with process in kell K_2 using channel a.*

Also, using extended semantics it is possible to encode both the LTS and reduction semantics into a single tool. In reduction semantics, only τ transitions are considered. In LTS semantics, abstraction and concretion transitions are considered, and τ transitions are not inspected for matching abstraction and concretion information.

4 Property Representation

Once a system has been modelled using kell-m, the operational semantics of kell-m can be used to study the evolution of the system. In particular, we are interested in being able to specify properties that must hold as a system evolves.

In this section we present kμ, a formalism for the specification of properties of systems represented using kell-m. kμ is a modal temporal logic based on the extended operational semantics for kell-m. For a given kell-m process, formulas specified in kμ impose conditions on the labelled transition system for the process. Conditions may specify kell containment requirements that must hold during transitions, and further kμ conditions that must hold after channel communications or kell passivations.

4.1 Syntax

Properties in kμ are formulas $\mathcal{F}$ with the syntax specified in Figure 12. The syntax is inspired by the language implementation of the $\pi\mu$-calculus in the Mobility Model Checker [17]. Along with the term kμ *formula*, we also use the term kμ *condition* when referring to a kμ property.

`tt` represents true, `ff` represents false, $\neg\mathcal{F}$ is used to negate a condition specified by a formula. a and b represent names, n represents a number. $\mathcal{C}$ specifies comparison of values passed in communications, containment conditions on lists of names, containment conditions on kell sets, or checks on the size of a kell containment set.

Diamond $\langle\varphi\rangle$ and box expressions $[\varphi]$ impose conditions on transitions, where φ is an action condition. Diamond expressions are used to specify *in at least one transition where φ* conditions; while box expressions are used to specify

in every transition where φ conditions. Hence, path quantification in kμ is implicit: existential for $\langle\ .\ \rangle$ modalities and universal for $[\ .\ ]$ modalities.

φ is a condition on a concretion or abstraction transition when φ is a pair (α, γ), and α represents an abstraction (i.e., $a(\widetilde{c})$, $K[x]$), or a concretion (i.e., $\overline{a}(\widetilde{w})$, $\overline{K}[P]$). γ represents a kell containment condition on the action α. Communication parameters in α (i.e., $\widetilde{c}$, $\widetilde{w}$, x, and P) can be variables and names. If they are names, they must match the names of the parameters in the transitions; if variables, they are instantiated with the corresponding parameter. For example, using uppercase for variables and lowercase for names, when α is $\overline{a}(n, w)$, the only matched transitions in the LTS are those labelled with output on channel a of names n and w. When α is $\overline{a}(V_1, V_2)$, the matched LTS transitions are those which output two values on channel a, irrespectively of the actual values output. More examples will be provided once we introduce kell containment conditions and the other φ expressions.

φ is a condition on a τ transition when φ is a triple $(\alpha_\tau, \gamma, \gamma)$, and α_τ represents τ actions (i.e., $\overleftrightarrow{a}(\widetilde{w})$, $\overleftrightarrow{K}[P]$). Recall from Section 3.4, $\overleftrightarrow{a}(\widetilde{w})$ specifies the matching of channel abstraction $a(\widetilde{c})$ and concretion $\overline{a}(\widetilde{w})$, and $\overleftrightarrow{K}[P]$ specifies the matching of kell abstraction $K[X]$ and concretion $\overline{K}[P]$. The first γ in the triple is the kell containment condition for the abstraction; the second γ is the kell containment condition for the matching concretion.

We refer to kell containment conditions as *any* for $*$; *exactly* for $\mathcal{K}$; *at least* for $\supseteq\ \mathcal{K}$; *except* for $\nsubseteq\ \mathcal{K}$; and *instantiation* for I, where I is a variable.

For an action α, kell containment condition $*$ does not impose any requirements on the location of the action. Let us now assume α is located in a process P such that:

$$K_1[K_2[\cdots K_n[P]\cdots]]$$

And α is not inside a kell in P. With $\mathcal{K}$, a set of kell names, $\mathcal{K}$ holds if $\{K_1, K_2, ..., K_n\} = \mathcal{K}$. For example, $\{K_1, K_2\}$ holds if α occurs in R with $K_1[K_2[R]]$ or $K_2[K_1[R]]$. The same condition does not hold in $K_3[K_1[K_2[R]]]$. $\supseteq\ \mathcal{K}$ holds if $\mathcal{K} \setminus \{K_1, K_2, ..., K_n\} = \emptyset$, and $\nsubseteq\ \mathcal{K}$ holds if $\mathcal{K} \cap \{K_1, K_2, ..., K_n\} = \emptyset$. If a variable I is specified as a kell containment condition, the condition succeeds and I is instantiated to $\{K_1, K_2, ..., K_n\}$.

Consider the LTS shown in Figure 13 for a process P. Because we are using kell-m's extended LTS semantics, for every label δ_i in the LTS, δ_i may be: a pair α_i, κ_i, with α_i an action and κ_i its kell containment set; or $(\alpha_\tau, \kappa_i, \kappa_i')$, with α_τ specifying the channel or kell involved in the communication, κ_i the kell containment set for the abstraction, and κ_i' the kell containment set for the concretion.

Let us assume a boolean function *cs*, able to decide if a sequence of parameters in α or α_τ expressions is compatible with the actual parameters of an action specified in a LTS transition. *cs* will be formally defined in Section 4.2. Intuitively, a sequence of parameters $\widetilde{w}$ in α or α_τ is compatible with the actual parameters $\widetilde{d}$ of an action if both sequences have the same number of values and, positionally, every name in $\widetilde{w}$ is matched in $\widetilde{d}$.

Condition $\langle\overline{a}(\widetilde{w}), \{T\}\rangle.\mathcal{F}$ holds for P if there is at least one transition from P labelled with communication action $\overline{a}(\widetilde{d})$, the action is located only within kell T, names in $\widetilde{w}$ match names in $\widetilde{d}$, $cs(\widetilde{w}, \widetilde{d})$, and, after the transition, the formula $\mathcal{F}\{\widetilde{d}/\widetilde{w}\}$ holds for the resulting process expression. Formally, $\langle\overline{a}(\widetilde{w}), \{T\}\rangle.\mathcal{F}$ holds if $\exists P_i : P \xrightarrow{\delta_i} P_i$, such that $\delta_i = (\alpha_i, \kappa_i)$, with $\alpha_i = \overline{a}(\widetilde{d})$, $cs(\widetilde{w}, \widetilde{d})$, $\kappa_i = \{T\}$, and $\mathcal{F}\{\widetilde{d}/\widetilde{w}\}$ holds at P_i.

Conditions on transitions representing abstractions are similarly specified. For example $\langle a(\widetilde{c}), \{T\}\rangle.\mathcal{F}$ holds if $\exists P_i : P \xrightarrow{\delta_i} P_i$, such that $\delta_i = (\alpha_i, \kappa_i)$, with $\alpha_i = \overline{a}(\widetilde{e})$, $cs(\widetilde{c}, \widetilde{e})$, $\kappa_i = \{T\}$, and $\mathcal{F}\{\widetilde{e}/\widetilde{c}\}$ holds at P_i.

A condition on a τ transition, for example $\langle\overleftrightarrow{a}(\widetilde{w}), \{K\}, \{T\}\rangle.\mathcal{F}$, holds if $\exists P_i : P \xrightarrow{\delta_i} P_i$, such that $\delta_i = \overleftrightarrow{a}(\widetilde{d}), \kappa_a, \kappa_c$, with $\kappa_a = \{K\}$, $cs(\widetilde{d}, \widetilde{w})$, $\kappa_c = \{T\}$, and $\mathcal{F}\{\widetilde{d}/\widetilde{w}\}$ holds at P_i. A τ action on a kell holds when the specified kell is passivated, for example because the process in the kell is being adapted. In such case a property specifying that a kell K is being passivated while executing in L_c by a process executing in L_a is written as follows:

$$\langle\overleftrightarrow{K}(X), \{L_a\}, \{L_c\}\rangle$$

Such a property holds for the following process:

$$L_c[Q \mid K[P]] \mid L_a[R \mid K[X] \triangleright K[P']]$$

$[\overline{a}(\widetilde{w}), \emptyset].\mathcal{F}$ holds if, for every transition labelled with action $\overline{a}(\widetilde{d})$ not located within any kell, $\mathcal{F}\{\widetilde{d}/\widetilde{w}\}$ holds for the process expression after the transition. Formally, $[\overline{a}(\widetilde{w}), \emptyset].\mathcal{F}$ holds when $\forall P_i : P \xrightarrow{\alpha_i, \kappa_i} P_i$, if $\alpha_i = \overline{a}(\widetilde{w})$, $cs(\widetilde{d}, \widetilde{w})$, and $\kappa_i = \emptyset$, then $\mathcal{F}$ holds at P_i.

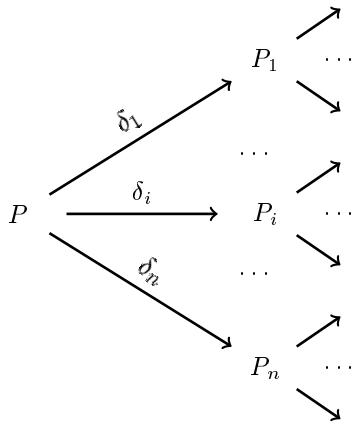

Figure 13: LTSs for $\langle\ .\ \rangle$ and $[\ .\ ]$ Examples

Similarly, a condition $[\overleftrightarrow{a}(\widetilde{w}), \{K\}, \emptyset].\mathcal{F}$, holds when $\forall P_i : P \xrightarrow{\delta_i} P_i$, if $\delta_i = [\overleftrightarrow{a}(\widetilde{d}), \kappa_a, \kappa_c]$, $cs(\widetilde{d}, \widetilde{w})$, and both, $\kappa_a = \{K\}$ and $\kappa_c = \{\}$, then $\mathcal{F}$ holds at P_i.

A property requiring adaptation of processes to occur only within certain localities is:

$$[\overleftrightarrow{K}(X), \{L_a\}, \{L_c\}]$$

If a kell K is passivated, the previous property requires the passivation to occur always while the passivated process executes within L_c by a process executing within L_a.

The use of variables as kell containment conditions is useful when requiring two different transitions to have the same kell containment conditions, without the need for knowing the actual kells. Hence $\langle\overline{a}(\widetilde{w}), I\rangle.\langle c(\widetilde{e}), I\rangle$ holds when there are consecutive transitions $P \xrightarrow{\overline{a}(\widetilde{d}),\kappa} Q \xrightarrow{c(\widetilde{g}),\kappa} R$. Notice the kell containment conditions κ are the same in both transitions.

The other possible $\langle\ .\ \rangle$ and $[\ .\ ]$ modalities deal with negation and sets of φ transition conditions. Having a process P and a set of communication condition pairs S, $\langle -\varphi\rangle.\mathcal{F}$ holds if there is at least one transition from P, for which φ does not stand, after which $\mathcal{F}$ holds. $\langle S\rangle.\mathcal{F}$ holds if there is at least one transition from P, for which a $\varphi \in S$ stands, after which $\mathcal{F}$ holds. $\langle -S\rangle.\mathcal{F}$ holds if there is at least one transition from P, for which a $\varphi \notin S$ stands, after which $\mathcal{F}$ holds. $[-\varphi].\mathcal{F}$ holds if $\mathcal{F}$ holds after every transition from P for which a φ' with $\varphi' \neq \varphi$ holds. $[S].\mathcal{F}$ holds if $\mathcal{F}$ holds after every transition from P for which $\varphi \in S$ holds. Similarly, $[-S].\mathcal{F}$ holds if $\mathcal{F}$ holds after every transition from P for which a condition $\varphi \notin S$ holds. We write $\langle -\rangle.\mathcal{F}$ for $\langle -\{\}\rangle.\mathcal{F}$, and $[-].\mathcal{F}$ for $[-\{\}].\mathcal{F}$.

Formulas can be named and parameterized using the syntax $N(\widetilde{x}) \stackrel{def}{=} \mathcal{F}$, where $\widetilde{x}$ are the formula parameters. $N(\widetilde{p})$ is an actual use of a named formula, where $\widetilde{p}$ are the values passed as formula parameters. Values can be channel and kell names, variables, and other named formulas. Naming of formulas allows recursive definitions. For example, the following formula has parameters C and K, and holds if, eventually, there is a read on given channel C within given kell K:

$$\mathit{ReadInKell}(C, K) \stackrel{def}{=} (\langle\overline{C}(\widetilde{c}), \supseteq \{K\}\rangle.\mathtt{tt}) \vee \langle -\rangle.\mathit{ReadInKell}(C, K)$$

Finally, notice there are no explicit quantifiers over variables. In the $\pi\mu$-calculus it is possible to express conditions $\forall c.\mathcal{F}$ requiring $\mathcal{F}$ to hold for every name c. In kμ variables are quantified implicitly depending on whether a diamond or box modality is used. Like paths, variables are existentially quantified in $\langle\ .\ \rangle$ and universally quantified in $[\ .\ ]$. For example, C is existentially quantified in $\langle a(C), *\rangle.\mathcal{F}$ and universally quantified in $[a(C), *].\mathcal{F}$. Since variable quantification is implicit in kμ, and only for values read or written, it is not possible to express conditions such as $\forall A : [A(C), *].\mathcal{F}$, meaning $\mathcal{F}$ *holds after every channel read*. However, variables can be used in place of channels and kells when the variables have been previously instantiated. For example, in the following property, variable *Rc* is instantiated with the third value passed in the communication on channel *subscribe*. The actual value corresponds to a channel on which a communication occurs right after the communication on channel *subscribe*:

$$\langle\overleftrightarrow{\mathit{subscribe}}(\mathit{Filter, Callback, Rc})\rangle.\langle\overleftrightarrow{\mathit{Rc}}(S)\rangle$$

P	$\models_{\mathcal{V}}$ $\mathtt{tt}$	always
P	$\models_{\mathcal{V}}$ $\neg\mathcal{F}$	if $P \not\models_{\mathcal{V}} \mathcal{F}$
P	$\models_{\mathcal{V}}$ $\mathcal{C}.\mathcal{F}$	if $\mathcal{C} \wedge P \models_{\mathcal{V}} \mathcal{F}$
P	$\models_{\mathcal{V}}$ $\mathcal{F}_1 \wedge \mathcal{F}_2$	if $P \models_{\mathcal{V}} \mathcal{F}_1 \wedge P \models_{\mathcal{V}} \mathcal{F}_2$
P	$\models_{\mathcal{V}}$ $\mathcal{F}_1 \vee \mathcal{F}_2$	if $P \models_{\mathcal{V}} \mathcal{F}_1 \vee P \models_{\mathcal{V}} \mathcal{F}_2$
P	$\models_{\mathcal{V}}$ $\langle\alpha_\tau, \gamma_a, \gamma_c\rangle.\mathcal{F}$	if $\exists Q : P \xrightarrow{\alpha'_\tau, \kappa_a, \kappa_c} Q \wedge \mathit{cmp}(\alpha_\tau, \alpha'_\tau) \wedge \mathit{kc}(\gamma_a, \kappa_a) \wedge \mathit{kc}(\gamma_c, \kappa_c) \wedge Q \models_{\mathcal{V}} \mathcal{F}'$
P	$\models_{\mathcal{V}}$ $\langle\alpha, \gamma\rangle.\mathcal{F}$	if $\exists Q : P \xrightarrow{\alpha', \kappa} Q \wedge \mathit{cmp}(\alpha, \alpha') \wedge \mathit{kc}(\gamma, \kappa) \wedge Q \models_{\mathcal{V}} \mathcal{F}''$
P	$\models_{\mathcal{V}}$ $[\alpha_\tau, \gamma_a, \gamma_c].\mathcal{F}$	if $\forall Q : P \xrightarrow{\alpha'_\tau, \kappa_a, \kappa_c} Q$, $(\mathit{cmp}(\alpha_\tau, \alpha'_\tau), (\mathit{kc}(\gamma_a, \kappa_a) \wedge \mathit{kc}(\gamma_c, \kappa_c))) \Rightarrow Q \models_{\mathcal{V}} \mathcal{F}'$
P	$\models_{\mathcal{V}}$ $[\alpha, \gamma].\mathcal{F}$	if $\forall Q : P \xrightarrow{\alpha', \kappa} Q$, $(\mathit{cmp}(\alpha, \alpha') \wedge \mathit{kc}(\gamma, \kappa)) \Rightarrow Q \models_{\mathcal{V}} \mathcal{F}''$
P	$\models_{\mathcal{V}}$ $\langle -(\alpha, \gamma)\rangle.\mathcal{F}$	if $\exists Q : P \xrightarrow{\alpha', \kappa} Q \wedge (\neg\mathit{cmp}(\alpha, \alpha') \vee \neg\mathit{kc}(\gamma, \kappa)) \wedge Q \models_{\mathcal{V}} \mathcal{F}'$, or if $\exists Q : P \xrightarrow{\alpha_\tau, \kappa_a, \kappa_c} Q \wedge Q \models_{\mathcal{V}} \mathcal{F}'$
P	$\models_{\mathcal{V}}$ $\langle -(\alpha_\tau, \gamma_a, \gamma_c)\rangle.\mathcal{F}$	if $\exists Q : P \xrightarrow{\alpha'_\tau, \kappa_a, \kappa_c} Q \wedge (\neg\mathit{cmp}(\alpha_\tau, \alpha'_\tau) \vee \neg\mathit{kc}(\gamma_a, \kappa_a) \vee \neg\mathit{kc}(\gamma_c, \kappa_c)) \wedge Q \models_{\mathcal{V}} \mathcal{F}'$, or if $\exists Q : P \xrightarrow{\alpha, \kappa} Q \wedge Q \models_{\mathcal{V}} \mathcal{F}'$
P	$\models_{\mathcal{V}}$ $\langle\{\varphi_1, \varphi_2, ..., \varphi_n\}\rangle.\mathcal{F}$	if $P \models_{\mathcal{V}} \langle\varphi_1\rangle.\mathcal{F} \vee P \models_{\mathcal{V}} \langle\varphi_2\rangle.\mathcal{F} \vee \cdots \vee P \models_{\mathcal{V}} \langle\varphi_n\rangle.\mathcal{F}$
P	$\models_{\mathcal{V}}$ $\langle -S\rangle.\mathcal{F}$	if $\exists\varphi : P \models_{\mathcal{V}} \langle\varphi\rangle.\mathcal{F} \wedge \varphi \notin S$
P	$\models_{\mathcal{V}}$ $[-\varphi].\mathcal{F}$	if $P \models_{\mathcal{V}} [-\{\varphi\}].\mathcal{F}$
P	$\models_{\mathcal{V}}$ $[\emptyset].\mathcal{F}$	always
P	$\models_{\mathcal{V}}$ $[\{\varphi_1, \varphi_2, ..., \varphi_n\}].\mathcal{F}$	if $P \models_{\mathcal{V}} [\varphi_1].\mathcal{F} \wedge P \models_{\mathcal{V}} [\varphi_2].\mathcal{F} \wedge \cdots \wedge P \models_{\mathcal{V}} [\varphi_n].\mathcal{F}$
P	$\models_{\mathcal{V}}$ $[-S].\mathcal{F}$	if $P \models_{\mathcal{V}} [\varphi].\mathcal{F}$, with $\varphi \notin S$

Figure 14: kμ Semantics

4.2 Semantics

For a kμ formula $\mathcal{F}$, an *interpretation* $\mathcal{V}$ of formula parameters is a total function $\mathcal{V} : V \to N_K \cup N_F \cup N_V$. V are the parameter names in $\mathcal{F}$, N_k are kell-m names (of channels and kells), N_F are named formulas, and N_V are property variables, including variables in kell containment conditions. For simplicity we will assume no name clashes: $N_K \cap N_F \cap N_V = \emptyset$. Given a process P, a formula $\mathcal{F}$, and interpretation $\mathcal{V}$ of parameters in $\mathcal{F}$, we write $P \models_{\mathcal{V}} \mathcal{F}$ when $\mathcal{F}$ holds in P. Based on the LTS for P according to the extended semantics of kell-m (cf. Section 3.4), $\models_{\mathcal{V}}$ is defined inductively in Figure 14.

Because of property variables in the α and γ expressions, once a transition has occurred the resulting formula needs to be alpha-converted according to the instantiation of variables. Two kinds of instantiations can occur, depicted as $\mathcal{F}'$ and $\mathcal{F}''$, both specified in Figure 15. $\mathcal{F}'$ is used for τ transitions, while $\mathcal{F}''$ is used for abstractions and concretions.

Auxiliary functions used in the specification of the kμ semantics are defined in Figure 16. *kc* is a boolean function that, given a kell containment condition γ and kell containment set κ, decides if the kell containment condition holds. *cmp* is a boolean function able to decide if a condition action in a property formula applies to an action in a LTS transition. *cmp* uses boolean formula *cs* in its definition. *cs* determines if a sequence of parameter names and variables specified in a transition condition is compatible with a sequence of values specified in a LTS transition action. Finally, overloaded function *ps* extracts the list of parameters in an action condition or an action in a LTS transition.

Fixed point conditions are expressed using recursive formulas. For example, one may be interested in specifying

$$\mathcal{F}' = \begin{cases} \mathcal{F}\{ps(\alpha'_\tau)/ps(\alpha_\tau)\} & \text{if } \gamma_a \notin N_v, \gamma_b \notin N_v \\ \mathcal{F}\{\kappa_a, ps(\alpha'_\tau)/\gamma_a, ps(\alpha_\tau)\} & \text{if } \gamma_a \in N_v, \gamma_b \notin N_v \\ \mathcal{F}\{\kappa_c, ps(\alpha'_\tau)/\gamma_c, ps(\alpha_\tau)\} & \text{if } \gamma_a \notin N_v, \gamma_b \in N_v \\ \mathcal{F}\{\kappa_a, \kappa_c, ps(\alpha'_\tau)/\gamma_a, \gamma_c, ps(\alpha_\tau)\} & \text{if } \gamma_a \in N_v, \gamma_b \in N_v \end{cases}$$

$$\mathcal{F}'' = \begin{cases} \mathcal{F}\{ps(\alpha')/ps(\alpha)\} & \text{if } \gamma \notin N_v \\ \mathcal{F}\{ps(\kappa, \alpha')/ps(\gamma, \alpha)\} & \text{if } \gamma \in N_v \end{cases}$$

Figure 15: Variable Instantiation in kμ Formulas

condition *eventually, there is a write on channel* a:

$$F \stackrel{def}{=} (\langle \overline{a}(\widetilde{w}), \texttt{any} \rangle . \texttt{tt}) \vee \langle - \rangle . F$$

Which corresponds to the following least fixed-point definition:

$$\mu y.(\langle \overline{a}(\widetilde{w}), \texttt{any} \rangle . \texttt{tt} \ \vee \ \langle - \rangle . y)$$

Similarly, an example of a greatest fixed-point is condition *it is always possible to read on channel* a:

$$A \stackrel{def}{=} \langle a(\widetilde{c}), (\texttt{any} \rangle . \texttt{tt}) \ \wedge \ \langle - \rangle . A$$

Which corresponds to:

$$\nu y.(\langle a(\widetilde{c}), \texttt{any} \rangle . \texttt{tt} \ \wedge \ \langle - \rangle . y)$$

Another least fixed-point condition is *deadlock*, specified as:

$$\mathit{deadlock} \stackrel{def}{=} [-] . \texttt{ff} \ \vee \ [-] . \mathit{deadlock}$$

An example of a deadlocked process is **new** $a\ a(c) \triangleright P$. This process is deadlocked because it is waiting for communication on a private channel a no other process knows.

5 Encoding kell-m

In this section we show how to encode kell-m into MMCπ. MMCπ is an extension of the π-calculus as specified in [1, 17] and implemented in the Mobility Model Checker (MMC). MMC is a tool for the verification of systems specified using MMCπ. MMC is written in XSB Prolog [2], a Prolog implementation with tabled evaluation [4]. Although not shown in the report, having an encoding of kell-m into MMCπ allows the development of a kell-m model checker using MMC. The syntax of MMCπ is as follows:

$$P \quad ::= \quad \mathbf{0} \ \mid \ \textbf{new}\ a\ P \ \mid \ P|P \ \mid \ P\text{+}P \ \mid \ \overline{a}(\widetilde{c}).P \ \mid \ a(\widetilde{c}).P \ \mid \ P(\widetilde{c}) \ \mid \ [a = b]P$$

$\widetilde{c}$ represents zero or more names. The *choice* operator $+$, represents a non-deterministic choice between two processes. $[a = b]P$ behaves as P if a and b are the same, otherwise it behaves as $\mathbf{0}$. $P(\widetilde{c})$ is process invocation, where P has been defined as $P(\widetilde{d}) \stackrel{def}{=} P_d$. Recursion is allowed in process definitions. Because recursion is allowed, there is no replication operation $!P$ (in the π-calculus, $P! \stackrel{def}{=} P \mid !P$, [8]).

To encode the kell-m calculus in MMCπ two main issues need to be addressed: the semantics of kells, and the higher-order nature of the kell-m calculus.

As established by the semantics of kell-m (cf. Figure 4, and Section 3.1), at a given point, a kell may be non-deterministically passivated by a trigger, or it may continue its execution. When encoding kell-m in MMCπ, we use the name of the kell as a communication channel where the kell's own process is output. For example, a kell $K[P]$ is represented as:

$$\mathit{choice}(\overline{K}(P), K[\mathit{advance}(P)])$$

Where *choice* behaves as the π-calculus non-deterministic choice between two processes, and *advance* advances the execution of a process.

$$
\begin{array}{rl}
kc(*, \kappa) & \text{always} \\
kc(\mathcal{K}, \kappa) & \text{if } (\kappa = \mathcal{K}) \\
kc(\supseteq \mathcal{K}, \kappa) & \text{if } (\mathcal{K} \setminus \kappa = \emptyset) \\
kc(\not\subseteq \mathcal{K}, \kappa) & \text{if } (\kappa \cap \mathcal{K} = \emptyset) \\
kc(V, \kappa) & \text{always}
\end{array}
$$

$$
\begin{array}{rl}
cmp(a(\widetilde{c}), g(\widetilde{e})) & \text{if } a = g \wedge cs(\widetilde{c}, \widetilde{e}) \\
cmp(\overline{a}(\widetilde{w}), \overline{g}(\widetilde{d})) & \text{if } a = g \wedge cs(\widetilde{w}, \widetilde{d}) \\
cmp(K[x], T[z]) & \text{if } K = T \wedge cs(x, z) \\
cmp(\overline{K}[x], \overline{T}[P]) & \text{if } K = T \wedge cs(x, P) \\
cmp(\overleftrightarrow{a}(\widetilde{w}), \overleftrightarrow{g}(\widetilde{d})) & \text{if } a = g \wedge cs(\widetilde{w}, \widetilde{d}) \\
cmp(\overleftrightarrow{K}[x], \overleftrightarrow{T}[P]) & \text{if } K = T \wedge cs(x, P)
\end{array}
$$

$$
\begin{array}{rl}
cs(\widetilde{w}, \widetilde{c}) & \text{if } |\widetilde{w}| = |\widetilde{d}| = 0 \\
cs(\widetilde{w}, \widetilde{c}) & \text{if } \widetilde{w} = w, \widetilde{w}' \wedge \widetilde{c} = c, \widetilde{c}' \wedge \\
 & (w \in \mathcal{V} \vee (w \in N_k \wedge w = c)) \wedge cs(\widetilde{w}', \widetilde{c}')
\end{array}
$$

$$
\begin{array}{rcl}
ps(\overline{a}(\widetilde{w})) & \equiv & \widetilde{w} \\
ps(a(\widetilde{c})) & \equiv & \widetilde{c} \\
ps(K[x]) & \equiv & x \\
ps(\overline{K}[P]) & \equiv & P \\
ps(\overleftrightarrow{a}(\widetilde{w})) & \equiv & \widetilde{w} \\
ps(\overleftrightarrow{K}[P]) & \equiv & P
\end{array}
$$

Figure 16: Auxiliary Functions in kμ Semantics Definition

5.1 Higher-Order Expressions

To deal with higher-order expressions in kell-m we could try Sangiorgi's approach, originally proposed to reduce higher-order π-calculus expressions to behaviour equivalent first-order π-calculus expressions [12, 13]. In Sangiorgi's approach, higher-order expressions are replaced by names we call *higher-order references*. Every higher-order expression is then composed in parallel and activated by a higher-order reference. The resulting first-order expression, could then be converted to a π-calculus expression.

Comparing with traditional programming languages, one can think of Sangiorgi's approach as the passing of function pointers instead of the function code itself. For example, consider the following higher-order kell-m process:

$$\overline{a}(\overline{b}(c)) \mid a(x) \triangleright (x \mid x)$$

The process is equivalent to a kell-m expression where $\overline{b}(c)$ is replaced with a fresh name h, and the higher-order expression is factorized into a process activated by channel h:

$$\textbf{new } h \ (\overline{a}(h) \mid h() \triangleright \overline{b}(c) \mid h() \triangleright \overline{b}(c)) \mid a(x) \triangleright (\overline{x}() \mid \overline{x}())$$

The fresh name h is, in essence, a reference to the factorized higher-order expression it replaced. Unfortunately, this approach is not compatible with kell semantics. Consider the following kell-m expression E_1, where P is a kell-m process:

$$E_1 \equiv \overline{a}(P) \mid a(x) \triangleright K[x]$$

Such an expressions is higher-order because process P is being output on channel a. Assuming Sangiorgi's approach applies to kell-m, such an expression would be equivalent to E_2:

$$E_2 \equiv \textbf{new } h \ (\overline{a}(h) \mid h() \triangleright P) | a(x) \triangleright K[\overline{x}()]$$

After there is a match between $\overline{a}(h)$ and $a(x)$ in E_2, kell K causes the activation of P by executing $\overline{x}()$. P then executes *outside* kell K. In the original expression $a(x) \triangleright K[x]$ the intention is to execute the received kell-m expression P *inside* kell K. Because P executes inside K, another process can use a trigger $K[y] \triangleright Q$ to suspend K in the middle of P's execution. In E_2 there is no way to suspend the execution of P once it has been activated via h. The behaviour in E_2 is that of remote code invocation, whilst the behaviour in E_1 is similar to code migration and local execution of the received code.

Another interesting feature related to the higher-order nature of the kell-m calculus is the scope extrusion of restricted names when kells are passivated. A process expression, being sent from one kell to another, may have restricted names that must remain restricted at the receiving kell. For example, in:

$$K_1[\textbf{new } c,d\ \overline{a}(\overline{c}(d)|c(e) \triangleright \mathbf{0})] \mid K_2[a(x) \triangleright x]$$

when $(\overline{c}(d)|c(e) \triangleright \mathbf{0})$ is received by K_2, names c and d should remain restricted. This scope extrusion is explicitly represented by the reduction rules *-REDUCTEXTRUSION presented in Section 3.2.

We call the *process context* of a higher-order process expression P, the set of restricted names for P at a given time. When there is code-shipping, the context of a given higher-order expression may vary depending on what code is actually received. For example, in:

$$\overline{a}(P) \mid \overline{a}(Q) \mid a(x) \triangleright a[x]$$

The context of the expression $a[x]$ depends on which one of P and Q is matched by $a(x)$. For some executions it may be P, for others it may be Q. Only until one analyses a specific execution it is possible to determine which context the expression $a[x]$ took during the execution.

When code is not migrated but referenced, as it is the case in Sangiorgi's approach, there is no scope extrusion since the higher-order code is factorized into a process expression that shares the same context as the writing expression: $\overline{a}(P)$ is converted into **new** h $(\overline{a}(h) \mid h() \diamond P)$. When $\overline{h}()$, P executes within the same context as $\overline{a}(h)$.

Because of the issues with higher-order expression inside kells, and the need to carry-over processes along with their contexts when they are transmitted via channels, we present a *runable* interpretation of kell-m expressions as MMCπ expressions. We call it *runable*, because the resulting MMCπ expression is equivalent to a specific execution of the kell-m expression. We will show that the MMC encoding of a kell-m process simulates the kell-m process.

Our solution to deal with higher-order expressions in the MMC interpretation, is to replace the higher-order expressions with fresh names we call *higher-order indicators*. When a higher-order indicator is received by a trigger, the higher-order indicator is *replaced* by its associated kell-m expression. For example, in the following process:

$$\overline{a}(\overline{b}(c)) \mid a(x) \triangleright x$$

$\overline{b}(c)$ is replaced with a fresh name h:

$$\overline{a}(h) \mid \mathcal{H}[\![\ a(x)\ ]\!] \triangleright x \rightarrow \mathcal{H}[\![\ x\{h/x\}\ ]\!] \equiv \overline{b}(c)$$

Fresh name h is a higher-order indicator, and its relation to $\overline{b}(c)$ is remembered. When there is a communication, and the process reduces to $x\{h/x\} \equiv h$, h is then replaced by $\overline{b}(c)$. We call this process *instantiation of higher-order indicators*, and use $\mathcal{H}[\![\ \cdot\ ]\!]$ for its representation.

5.2 Fresh Names and Higher-Order Mappings

Before we define the MMCπ runable interpretation of kell-m processes, we introduce some auxiliary definitions. $\mathcal{F}[\![\ P\ ]\!]$ is defined in Figure 17. When Prolog variables are used care must be taken with expressions such as:

$$\textbf{new } c\ ((\textbf{new } c\ \overline{a}(c)) \mid \overline{b}(c))$$

Notice the c in $\overline{a}(c)$ is a different name than the c in $\overline{b}(c)$. If only one Prolog variable is used to represent c, when either $\overline{a}(c)$ or $\overline{b}(c)$ is instantiated, for example as part of a communication, the other c is instantiated as well. This problem is avoided by guaranteeing MMCπ expressions passed to MMC do not have name collisions.

Assuming H, the set of higher-order mappings in process expressions (initially empty), $\mathcal{H}[\![\ P\ ]\!]$ is defined as the instantiation of the higher-order indicators in a kell-m process P:

$$\mathcal{H}[\![\ P\ ]\!] \quad \equiv \quad P\{\widetilde{w}/\widetilde{h}\} \text{ with } |\widetilde{w}| = |\widetilde{h}|, \text{ and } \begin{cases} w_i = p_i, & \text{if } \exists p_i : (h_i, p_i) \in H \\ w_i = h_i, & \text{otherwise} \end{cases}$$

$\mathcal{H}$ is defined recursively in Figure 18.

$$
\begin{array}{rcl}
\mathcal{F}[\![\ \mathbf{0}\ ]\!] & \stackrel{def}{=} & \mathbf{0} \\
\mathcal{F}[\![\ x\ ]\!] & \stackrel{def}{=} & x \\
\mathcal{F}[\![\ \mathbf{new}\ a\ P\ ]\!] & \stackrel{def}{=} & \mathbf{new}\ v\ \mathcal{F}[\![\ P\{v/a\}\ ]\!] \text{ with } v \text{ a fresh name} \\
\mathcal{F}[\![\ \overline{a}(\widetilde{w})\ ]\!] & \stackrel{def}{=} & \overline{a}(\mathcal{F}[\![\ \widetilde{w}\ ]\!]) \\
\mathcal{F}[\![\ \widetilde{w}\ ]\!] & \stackrel{def}{=} & \mathcal{F}[\![\ w_1\ ]\!], ..., \mathcal{F}[\![\ w_n\ ]\!] \text{ where } \widetilde{w} \equiv w_1, ..., w_n \\
\mathcal{F}[\![\ K[P]\ ]\!] & \stackrel{def}{=} & K[\mathcal{F}[\![\ P\ ]\!]] \\
\mathcal{F}[\![\ a(\widetilde{c}) \triangleright P\ ]\!] & \stackrel{def}{=} & a(\widetilde{v}) \triangleright \mathcal{F}[\![\ P\{\widetilde{v}/\widetilde{c}\}\ ]\!] \text{ with } |\widetilde{v}| = |\widetilde{c}| \text{ and } v_i \in \widetilde{v} \text{ all fresh names} \\
\mathcal{F}[\![\ K[x] \triangleright P\ ]\!] & \stackrel{def}{=} & K[v] \triangleright \mathcal{F}[\![\ P\{v/x\}\ ]\!] \text{ with } v \text{ a fresh name} \\
\mathcal{F}[\![\ P \mid Q\ ]\!] & \stackrel{def}{=} & \mathcal{F}[\![\ P\ ]\!] \mid \mathcal{F}[\![\ Q\ ]\!] \\
\mathcal{F}[\![\ P(\widetilde{w})\ ]\!] & \stackrel{def}{=} & P(\mathcal{F}[\![\ \widetilde{w}\ ]\!])
\end{array}
$$

Figure 17: Introduction of Fresh Names

$$
\begin{array}{rcl}
\mathcal{H}[\![\ \mathbf{0}\ ]\!] & \stackrel{def}{=} & \mathbf{0} \\
\mathcal{H}[\![\ h\ ]\!] & \stackrel{def}{=} & \begin{cases} Q & \text{if } (h, Q) \in H \\ h & \text{otherwise} \end{cases} \\
\mathcal{H}[\![\ \mathbf{new}\ a\ P\ ]\!] & \stackrel{def}{=} & \mathbf{new}\ a\ \mathcal{H}[\![\ P\ ]\!] \\
\mathcal{H}[\![\ a(\widetilde{b}) \triangleright P\ ]\!] & \stackrel{def}{=} & a(\widetilde{b}) \triangleright \mathcal{H}[\![\ P\ ]\!] \\
\mathcal{H}[\![\ K[x] \triangleright P\ ]\!] & \stackrel{def}{=} & K[x] \triangleright \mathcal{H}[\![\ P\ ]\!] \\
\mathcal{H}[\![\ K[P]\ ]\!] & \stackrel{def}{=} & K[\mathcal{H}[\![\ P\ ]\!]] \\
\mathcal{H}[\![\ \overline{a}(w_1, ..., w_n)\ ]\!] & \stackrel{def}{=} & \overline{a}(w_1, ..., w_n) \\
\mathcal{H}[\![\ P(\widetilde{w})\ ]\!] & \stackrel{def}{=} & \mathcal{H}[\![\ P_d\{\widetilde{w}/\widetilde{y}\}\ ]\!] \text{ with } P(\widetilde{y}) \stackrel{def}{=} P_d \\
\mathcal{H}[\![\ P \mid Q\ ]\!] & \stackrel{def}{=} & \mathcal{H}[\![\ P\ ]\!] \mid \mathcal{H}[\![\ Q\ ]\!]
\end{array}
$$

Figure 18: Instantiation of Higher-Order Indicators

5.3 Kell Passivation

To implement kell-m extended semantics, it is necessary to keep track of kell containment information. A set K_s of kells is used for this purpose. When a kell-m abstraction or concretion is to be encoded in MMC, the input or output parameters are extended with the kell containment information.

A kell $K[P]$ can be passivated by a matching trigger $K[x] \triangleright Q$, or it can advance its execution. The choice between passivating the kell and advancing the execution of its process is non-deterministic. A kell $K[P]$ itself may be nested within another kell-m process. For example, consider $R = K_1[K_2[K[P]]]$.

We specify $\mathcal{S}[\![\ R, B, K[P], K_s\ ]\!]$ as the MMCπ process corresponding to the passivation of a kell $K[P]$ nested in process R, executing within kells K_s, and with restricted names B (cf. Figure 19).

Hwrite replaces higher-order expressions in a channel output with higher-order indicators. For example, *Hwrite*$(\overline{a}(\overline{b}(c).\mathbf{0}).\mathbf{0})$ produces the expression **new** h $(\overline{a}(h).\mathbf{0})$, where $H := H \cup \{(h, \overline{b}(c))\}$.

$\mathcal{I}[\![\ B, P\ ]\!]$, to be defined later, is the MMC π-calculus interpretation of P when B represents the set of restricted names at the time P is interpreted.

Since higher-order process expressions can be output on channels, we need to include the restricted names in the process expressions as well. As previously mentioned, we refer to these restricted names, as the *context* of the process expressions output. Here we are abusing the notation by writing the set of kells K_s and restricted names B directly. Alternatively, one can assume an arbitrarily large number of names being passed in the concretions and being received in the abstractions. The first names would correspond to the elements of K_s and B, the rest of the names would be special names used to specify when a position is not being used. For notational convenience, we specify the set of kells and bound names directly. We extend this abuse of notation to kell-m channel abstractions and concretions. Therefore, we write $\overline{a}(\widetilde{w}, K_s)$ and $a(\widetilde{c}, K_s)$, with K_s kell containment sets.

Kell-m concretions $\overline{a}(\widetilde{w})$ are encoded as MMCπ expressions $\overline{a}(\widetilde{w}, K_a, K_c, B)$. As we will show later, abstractions $a(\widetilde{c})$ are encoded as $a(\widetilde{c}, K_a, K_c, B)$. If a concretion, K_c is the kell containment set for the action and K_a is a Prolog variable which is instantiated at the time of a τ transition involving the channel a. If an abstraction, K_a is the kell

$$
\begin{aligned}
\mathcal{S}[\![\, R, B, K[P], K_s \,]\!] &\stackrel{def}{=} \mathit{Hwrite}(\overline{K}(P, K_s), \mathcal{I}[\![\, B, R\{\mathbf{0}/K[P]\} \,]\!], B \setminus \{K\}) \\
&\quad + \begin{cases} \mathcal{S}[\![\, R, B, P, K_s \cup \{K\} \,]\!] & \text{if } R \neq \mathbf{0} \\ \mathcal{S}[\![\, K[P], B, P, K_s \cup \{K\} \,]\!] & \text{otherwise} \end{cases} \\
\mathcal{S}[\![\, R, B, P \mid Q, K_s \,]\!] &\stackrel{def}{=} \mathcal{S}[\![\, R, B, P, K_s \,]\!] \;+\; \mathcal{S}[\![\, R, B, Q, K_s \,]\!] \\
\mathcal{S}[\![\, R, B, P, K_s \,]\!] &\stackrel{def}{=} \mathbf{0} \text{ if } P \neq Q \mid R, \text{ and } P \neq K[Q] \\
\mathit{Hwrite}(\overline{a}(\widetilde{w}, K_s), Q, B) &\stackrel{def}{=} \mathit{Hconv}(\overline{a}(\widetilde{w}, K_a, K_s, B).Q, |\widetilde{w}|, 1), \text{ with } K_a \text{ a variable}
\end{aligned}
$$

$$
\mathit{Hconv}(P, n, i) \stackrel{def}{=} \begin{cases} P & \text{if } i \geq n \\ \mathit{Hconv}(\mathbf{new}\ h\ (P\{h/w_i\}), n, i+1), & \text{with } h \text{ fresh and} \\ & H := H \cup \{(h, w_i)\} \\ & \text{if } w_i \text{ is not a name} \\ \mathit{Hconv}(P, n, i+1) & \text{otherwise} \end{cases}
$$

Figure 19: Encoding of Kells

containment set, and K_c and B are variables instantiated at τ transitions. Hence, we are taking advantage of Prolog's unification for exposing kell containment information of both, concretions and abstractions.

Consider the kell-m process:

$$K_1[K_2[K_3[a(\widetilde{c}) \triangleright \mathbf{0}] \mid \overline{a}(\widetilde{w})]]$$

$\mathcal{S}$ generates MMCπ code allowing the passivation of any of the three kells. Let $Q = K_2[K_3[a(\widetilde{c}) \triangleright \mathbf{0}] \mid \overline{a}(\widetilde{w})]$, $R = K_3[a(\widetilde{c}) \triangleright \mathbf{0}] \mid \overline{a}(\widetilde{w})$, and $P = K_1[Q]$. By $\mathcal{S}$'s definition, the process is encoded as shown in Figure 20. In the resulting process expression, the passivated kell, if any, is replaced by the null process in P. If τ transitions, K_a, K'_a, and K''_a in Figure 20, are variables to be instantiated with kell containment sets of matching kell passivation triggers.

5.4 Advancing Kell Processes

If a kell $K[P]$ is not passivated, its process advances its execution as defined by $\mathcal{A}[\![\, K[P] \,]\!]$ in Figure 21(a). As mentioned at the beginning of Section 5 kell abstractions are converted to channel abstractions. For concretions and abstractions on channels ($\overline{a}(\widetilde{w})$, $a(\widetilde{c})$), and abstractions on kells ($K[x]$), a kell containment set, initially empty, is added to the parameters of the communication. For kell concretions ($K[P]$), the kell containment set is added when the code allowing the kell passivation is generated by $\mathcal{S}[\![\, . \,]\!]$.

Notice the re-invocation of $\mathcal{A}$ in the definition for $\mathcal{A}[\![\, K[L[P]] \,]\!]$. The other, alternate but incorrect, definition is:

$$\mathcal{A}[\![\, K[L[P]] \,]\!] \quad \stackrel{def}{=} \quad K[\mathcal{A}[\![\, L[P] \,]\!]]$$

In the case of nested kells, this definition may cause the interruption of one of the kells before the execution has advanced. In other words, we want to lift the abstraction and concretion operations from nested kells in one single step. For example, consider the process:

$$K_1[K_2[K_3[\overline{a}(\widetilde{w})]]]$$

By $\mathcal{A}$'s definition,

$$\mathcal{A}[\![\, K_1[K_2[K_3[\overline{a}(\widetilde{w})]]] \,]\!] \equiv \overline{a}(\widetilde{w}, \{K_1, K_2, K_3\})$$

If the alternate incorrect definition for $\mathcal{A}$ is used, we obtain:

$$\mathcal{A}[\![\, K_1[K_2[K_3[\overline{a}(\widetilde{w})]]] \,]\!] \equiv K_1[K_2[\overline{a}(\widetilde{w}, \{K_3\})]]$$

As we will see later, when we define $\mathcal{I}$, this alternate definition would allow the passivation of kells K_1 and K_2 at this point, at which time the kell processes do not correspond to the original ones ($K_3[\overline{a}(\widetilde{w})]$ in the case of K_2, and $K_2[K_3[\overline{a}(\widetilde{w})]]$ in the case of K_1).

Assuming no name collisions, $\mathcal{A}[\![\, K[P|Q] \,]\!]$ is defined in Figure 21(b). The assumption of no name collisions is important to avoid unintended capturing of names when lifting the name restrictions outside the kells ($\mathcal{A}[\![\, K[\mathbf{new}\ c\ P \mid Q] \,]\!]$).

Notice there is no definition of $\mathcal{A}[\![\, x \,]\!]$, with x a process variable. This is because the execution of a kell process can only advance when process variables have been replaced by their corresponding process expressions.

$$
\begin{aligned}
\mathcal{S}[\![\, \mathbf{0}, B, P, K_s \,]\!] \;&\equiv\; \mathit{Hwrite}(\overline{K_1}(Q, K_s), \mathcal{I}[\![\, B, \mathbf{0} \,]\!], B \setminus \{K_1\}) \;+\; \mathcal{S}[\![\, P, B, Q, K_s \cup \{K_1\} \,]\!] \\
&\equiv\; \textbf{new}\ h_1\ \overline{K_1}(h_1, K_s, K_a, B \setminus \{K_1\}).\mathcal{I}[\![\, B, \mathbf{0} \,]\!] \\
&\quad + \mathit{Hwrite}(\overline{K_2}(R, K_s \cup \{K1\}), \mathcal{I}[\![\, B, K_1[\mathbf{0}] \,]\!], B \setminus \{K_2\}) \\
&\quad + \mathcal{S}[\![\, P, B, R, K_s \cup \{K_1, K_2\} \,]\!], \text{ with } H := H \cup \{(h_1, Q)\} \\
&\equiv\; \textbf{new}\ h_1\ \overline{K_1}(h_1, K_s, K_a, B \setminus \{K_1\}).\mathcal{I}[\![\, B, \mathbf{0} \,]\!] \\
&\quad + \textbf{new}\ h_2\ \overline{K_2}(h_2, K_s \cup \{K_1\}, K'_a, B \setminus \{K_2\}).\mathcal{I}[\![\, B, K_1[\mathbf{0}] \,]\!] \\
&\quad + \mathcal{S}[\![\, P, B, K_3[a(\widetilde{c}) \triangleright \mathbf{0}], K_s \cup \{K_1, K_2\} \,]\!] \\
&\quad + \mathcal{S}[\![\, P, B, \overline{a}(\widetilde{w}), K_s \cup \{K_1, K_2\} \,]\!], \\
&\quad \text{with } H := H \cup \{(h_2, R)\} \\
&\equiv\; \textbf{new}\ h_1\ \overline{K_1}(h_1, K_s, K_a, B \setminus \{K_1\}).\mathcal{I}[\![\, B, \mathbf{0} \,]\!] \\
&\quad + \textbf{new}\ h_2\ \overline{K_2}(h_2, K_s \cup \{K_1\}, K'_a, B \setminus \{K_2\}).\mathcal{I}[\![\, B, K_1[\mathbf{0}] \,]\!] \\
&\quad + \mathit{Hwrite}(\overline{K_3}(a(\widetilde{c}, K_s \cup \{K_1, K_2\}) \triangleright \mathbf{0}), \mathcal{I}[\![\, B, K_1[K_2[\mathbf{0} \mid \overline{a}(\widetilde{w})]] \,]\!], \\
&\qquad\qquad B \setminus \{K_3\}) \\
&\quad + \mathcal{S}[\![\, P, B, a(\widetilde{c}) \triangleright \mathbf{0}, K_s \cup \{K_1, K_2, K_3\} \,]\!] \;+\; \mathbf{0} \\
&\equiv\; \textbf{new}\ h_1\ \overline{K_1}(h_1, K_s, K_a, B \setminus \{K_1\}).\mathcal{I}[\![\, B, \mathbf{0} \,]\!] \\
&\quad + \textbf{new}\ h_2\ \overline{K_2}(h_2, K_s \cup \{K_1\}, K'_a, B \setminus \{K_2\}).\mathcal{I}[\![\, B, K_1[\mathbf{0}] \,]\!] \\
&\quad + \textbf{new}\ h_3\ \overline{K_3}(h_3, K_s \cup \{K_1, K_2\}, K''_a, B \setminus \{K_3\}). \\
&\qquad\qquad \mathcal{I}[\![\, B, K_1[K_2[\mathbf{0} \mid \overline{a}(\widetilde{w})]] \,]\!] \\
&\quad + \mathcal{S}[\![\, P, B, a(\widetilde{c}) \triangleright \mathbf{0}, K_s \cup \{K_1, K_2, K_3\} \,]\!] \;+\; \mathbf{0}, \\
&\quad \text{with } H := H \cup \{(h_3, a(\widetilde{c}) \triangleright \mathbf{0})\} \\
&\equiv\; \textbf{new}\ h_1\ \overline{K_1}(h_1, K_s, K_a, B \setminus \{K_1\}).\mathcal{I}[\![\, B, \mathbf{0} \,]\!] \\
&\quad + \textbf{new}\ h_2\ \overline{K_2}(h_2, K_s \cup \{K_1\}, K'_a, B \setminus \{K_2\}).\mathcal{I}[\![\, B, K_1[\mathbf{0}] \,]\!] \\
&\quad + \textbf{new}\ h_3\ \overline{K_3}(h_3, K_s \cup \{K_1, K_2\}, K''_a, B \setminus \{K_3\}). \\
&\qquad\qquad \mathcal{I}[\![\, B, K_1[K_2[\mathbf{0} \mid \overline{a}(\widetilde{w})]] \,]\!] \\
&\quad + \mathbf{0} \;+\; \mathbf{0}
\end{aligned}
$$

Figure 20: Sample Encoding of Nested Kells

5.5 MMCπ Calculus Interpretation of kell-m Processes

The runable MMC π-calculus interpretation of a kell-m process P_k is defined as:

$$\mathit{Interpret}(P_k) \stackrel{\mathit{def}}{=} \mathcal{I}[\![\, \{\}, \mathcal{F}[\![\, P_k \,]\!] \,]\!], \text{where } \mathit{fn}(P_k) = \emptyset$$

fn is the set of free names in a process. $\mathcal{F}[\![\, P \,]\!]$ is process P with fresh names. $\mathcal{I}[\![\, B, P \,]\!]$ is the MMC π-calculus interpretation of P when B represents the set of restricted names at the time P is interpreted.

$\mathcal{I}[\![\, B, P \,]\!]$ is defined in Figure 22. By passing the context along with the process expressions (see the definition for $\mathcal{I}[\![\, B, \overline{a}(\widetilde{w}) \,]\!]$ above) we guarantee any restricted names in the process expressions remain restricted at the receiving end of the communication. To avoid unintended capture of free names on the receiving process, $\mathcal{F}$ is used every time a process is invoked and every time there is communication.

Similarly to the definition of $\mathcal{A}$, kell containment sets are added for concretions and abstractions on channels ($\overline{a}(\widetilde{w})$, $a(\widetilde{c})$), and for abstractions on kells ($K[x]$). In our tool $\emptyset$ is represented as an empty Prolog list [ ].

Recall *Hwrite* replaces higher-order expressions in a channel output with higher-order indicators. The resulting MMCπ expression corresponds to the channel output with higher-order expressions replaced by higher-order indicator names, plus the process scope, followed by a given MMCπ (**0** in the definition for $\mathcal{I}[\![\, B, \overline{a}(\widetilde{w}) \,]\!]$).

It is in the definition for $\mathcal{I}[\![\, B, a(\widetilde{c}) \triangleright P \,]\!]$, that we make use of the MMC π-calculus extension `code(Op, P)`. As previously mentioned, process expression `code(Op, P)` performs the Prolog predicate `Op`, and then behaves like `P`. In the case of the kell-m encoding, `code(inst(`P`,` $\widetilde{c}$`,` *bnd*`,` P_π`),`P_π`)`, the arguments received in the channel (if any), are used in the replacement of higher-order indicators.

When the process being encoded is the parallel composition of two processes within a kell ($\mathcal{I}[\![\, B, K[Q|R] \,]\!]$), we assume, non-deterministically, either one can advance. Hence we are using extensional (interleaved) concurrency: we model concurrent behaviour by, non-deterministically, interleaving the actions of parallel processes [6].

$$
\begin{array}{rcl}
\mathcal{A}[\![\, K[\mathbf{0}] \,]\!] & \stackrel{def}{=} & \mathbf{0} \\
\mathcal{A}[\![\, K[\mathbf{new}\ c\ P] \,]\!] & \stackrel{def}{=} & \mathbf{new}\ c\ \mathcal{A}[\![\, K[P] \,]\!] \\
\mathcal{A}[\![\, K[\overline{a}(\widetilde{w})] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[\overline{a}(\widetilde{w}, \emptyset)] \,]\!] \\
\mathcal{A}[\![\, K[\overline{a}(\widetilde{w}, K_s)] \,]\!] & \stackrel{def}{=} & \overline{a}(\widetilde{w}, K_s \cup \{K\}) \\
\mathcal{A}[\![\, K[a(\widetilde{c}) \triangleright P] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[a(\widetilde{c}, \emptyset) \triangleright P] \,]\!] \\
\mathcal{A}[\![\, K[a(\widetilde{c}, K_s) \triangleright P] \,]\!] & \stackrel{def}{=} & a(\widetilde{c}, K_s \cup \{K\}) \triangleright K[P] \\
\mathcal{A}[\![\, K[L[x] \triangleright P] \,]\!] & \stackrel{def}{=} & L(x, \emptyset) \triangleright K[P] \\
\mathcal{A}[\![\, K[L[P]] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[\mathcal{A}[\![\, L[P] \,]\!]] \,]\!] \\
\mathcal{A}[\![\, K[P(\widetilde{w})] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[P_d\{\widetilde{w}/\widetilde{y}\}] \,]\!] \text{ where } P(\widetilde{y}) \stackrel{def}{=} P_d
\end{array}
$$

(a) Simple Process

$$
\begin{array}{rcl}
\mathcal{A}[\![\, K[\mathbf{0} \mid Q] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[Q] \,]\!] \\
\mathcal{A}[\![\, K[\mathbf{new}\ c\ P \mid Q] \,]\!] & \stackrel{def}{=} & \mathbf{new}\ c\ \mathcal{A}[\![\, K[P \mid Q] \,]\!] \\
\mathcal{A}[\![\, K[\overline{a}(\widetilde{w}) \mid Q] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[\overline{a}(\widetilde{w}, \emptyset) \mid Q] \,]\!] \\
\mathcal{A}[\![\, K[\overline{a}(\widetilde{w}, K_s) \mid Q] \,]\!] & \stackrel{def}{=} & \overline{a}(\widetilde{w}, K_s \cup \{K\}) \mid K[Q] \\
\mathcal{A}[\![\, K[(a(\widetilde{c}) \triangleright P) \mid Q] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[(a(\widetilde{c}, \emptyset) \triangleright P) \mid Q] \,]\!] \\
\mathcal{A}[\![\, K[(a(\widetilde{c}, K_s) \triangleright P) \mid Q] \,]\!] & \stackrel{def}{=} & a(\widetilde{c}, K_s \cup \{K\}) \triangleright K[P \mid Q] \\
\mathcal{A}[\![\, K[(L[x] \triangleright P) \mid Q] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[(L(x, \emptyset) \triangleright P) \mid Q] \,]\!] \\
\mathcal{A}[\![\, K[L[P] \mid Q] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[\mathcal{A}[\![\, L[P] \,]\!] \mid Q] \,]\!] \\
\mathcal{A}[\![\, K[P(\widetilde{w}) \mid Q] \,]\!] & \stackrel{def}{=} & \mathcal{A}[\![\, K[P_d\{\widetilde{w}/\widetilde{y}\} \mid Q] \,]\!] \text{ where } P(\widetilde{y}) \stackrel{def}{=} P_d
\end{array}
$$

(b) Composed Process

Figure 21: Advancing the Execution of a Non-passivated Kell

Notice there is a definition for $\mathcal{I}[\![\, B, x \,]\!]$, with x a process variable. A MMCπ expression cannot be produced for a process variable until the variable has been instantiated with a higher-order indicator, and the higher-order indicator has been replaced by its associated process expression. Higher-order indicators are determined when the writer and receiver processes communicate on a channel, via a τ transition. If higher-order indicators associated to process variables are not known, the result of the interpretation is the null process. For example, the kell-m expression $a(x) \triangleright x$ is encoded as $a(x).\mathbf{0}$.

We now need to show for a kell-m process P: (a) the result of $\mathcal{I}[\![\, \{\}, P \,]\!]$, is a MMC π-calculus process (cf. 5.6); and (b) a global observer cannot distinguish P from $\mathcal{I}[\![\, \{\}, P \,]\!]$.

5.6 $\mathcal{I}[\![\, \{\}, P \,]\!]$ is a MMCπ Expression

As in Section 5, we will abuse the notation and write $\overline{a}(\widetilde{w}, K_s)$ and $a(\widetilde{c}, K_s)$, with K_s kell containment sets. Moreover, we will consider these expressions valid kell-m expressions, even though kell containment sets should be represented using sequences of names.

Recall a MMCπ expression is a π-calculus expression extended with `code(Op, P)`. Both names and variables can be transmitted in communications. We will use induction on the structure of P and assume no name collisions in P (i.e., $\mathcal{F}$ has been invoked). If $P \equiv \mathbf{0}$, by definition of $\mathcal{I}$, $\mathcal{I}[\![\, B, \mathbf{0} \,]\!] \stackrel{def}{=} \mathbf{0}$, and $\mathbf{0}$ is trivially a MMCπ process. Similarly, if P is a process variable x, by definition of $\mathcal{I}$, $\mathcal{I}[\![\, B, x \,]\!] \stackrel{def}{=} \mathbf{0}$, and $\mathbf{0}$ is trivially a MMCπprocess. The other possible cases for P are:

- **new** $a\ Q$, by definition $\mathcal{I}[\![\, B, \mathbf{new}\ a\ Q \,]\!] \stackrel{def}{=} \mathbf{new}\ a\ \mathcal{I}[\![\, B \cup \{a\}, Q \,]\!]$. By induction $\mathcal{I}[\![\, B, Q \,]\!]$ is MMCπ. $\mathcal{I}[\![\, B \cup \{a\}, Q \,]\!]$ is also a MMCπ because, $\mathcal{I}$ does not impose conditions on B, and adding a name to the set of restricted names B, only affects the tuple of names input and output during channel actions (see the $\mathcal{I}$ definitions for $\overline{a}(\widetilde{w})$ and $a(\widetilde{c}) \triangleright R$ above). Finally, since restriction of a MMCπ is a MMCπ, then $\mathcal{I}[\![\, B, P \,]\!]$ is a MMCπ.

$$
\begin{array}{rcl}
\mathcal{I}[\![\, B, \mathbf{0} \,]\!] & \stackrel{def}{=} & \mathbf{0} \\
\mathcal{I}[\![\, B, x \,]\!] & \stackrel{def}{=} & \mathbf{0} \\
\mathcal{I}[\![\, B, \mathbf{new}\ a\ P \,]\!] & \stackrel{def}{=} & \mathbf{new}\ a\ \mathcal{I}[\![\, B \cup \{a\}, P \,]\!] \\
\mathcal{I}[\![\, B, \overline{a}(\widetilde{w}) \,]\!] & \stackrel{def}{=} & \mathcal{I}[\![\, B, \overline{a}(\widetilde{w}, \emptyset) \,]\!] \\
\mathcal{I}[\![\, B, \overline{a}(\widetilde{w}, K_s) \,]\!] & \stackrel{def}{=} & \mathit{Hwrite}(\overline{a}(\widetilde{w}, K_s), \mathbf{0}, B \setminus \{a\}) \\
\mathcal{I}[\![\, B, a(\widetilde{c}) \triangleright P \,]\!] & \stackrel{def}{=} & \mathcal{I}[\![\, B, a(\widetilde{c}, \emptyset) \triangleright P \,]\!] \\
\mathcal{I}[\![\, B, a(\widetilde{c}, K_s) \triangleright P \,]\!] & \stackrel{def}{=} & a(\widetilde{c}, K_s, K_c, \mathit{bnd}).\mathtt{code}(\mathtt{inst}(P, \widetilde{c}, \mathit{bnd}, P_\pi), P_\pi) \\
 & & \text{with } \mathtt{inst}(P, \widetilde{c}, \mathit{bnd}, P_\pi) \ \mathtt{:-}\ P_\pi = \mathcal{I}[\![\, B \cup \mathit{bnd}, \mathcal{F}[\![\, \mathcal{H}[\![\, P \,]\!] \,]\!] \,]\!] \\
 & & \text{and } K_c \text{ a variable} \\
\mathcal{I}[\![\, B, K[P] \,]\!] & \stackrel{def}{=} & \mathcal{S}[\![\, \mathbf{0}, B, K[P], \emptyset \,]\!] \\
 & & + \begin{cases} \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q|R] \,]\!] \,]\!] + \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[R|Q] \,]\!] \,]\!] & \text{if } P \equiv Q|R \\ \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[P] \,]\!] \,]\!] & \text{otherwise} \end{cases} \\
\mathcal{I}[\![\, B, K[x] \triangleright P \,]\!] & \stackrel{def}{=} & \mathcal{I}[\![\, B, K(x, \emptyset) \triangleright P \,]\!] \\
\mathcal{I}[\![\, B, P(\widetilde{w}) \,]\!] & \stackrel{def}{=} & \mathcal{I}[\![\, B, \mathcal{F}[\![\, P_d\{\widetilde{w}/\widetilde{y}\} \,]\!] \,]\!] \text{ where } P(\widetilde{y}) \stackrel{def}{=} P_d \\
\mathcal{I}[\![\, B, P \mid Q \,]\!] & \stackrel{def}{=} & \mathcal{I}[\![\, B, P \,]\!] \mid \mathcal{I}[\![\, B, Q \,]\!]
\end{array}
$$

Figure 22: Encoding of kell-m Processes into MMC's π-calculus

- $\overline{a}(\widetilde{w})$, by definition $\mathcal{I}[\![\, B, \overline{a}(\widetilde{w}) \,]\!] \stackrel{def}{=} \mathit{Hwrite}(\overline{a}(\widetilde{w}, \emptyset), \mathbf{0}, B')$, with $B' = B \cup \{a\}$. Since $\mathit{Hwrite}(\overline{a}(\widetilde{w}, \emptyset), \mathbf{0}, B') \stackrel{def}{=} \mathit{Hconv}(\overline{a}(\widetilde{w}, K_a, \emptyset, B').\mathbf{0}, |\widetilde{w}|, 1)$ with K_a a variable, we need to show $\mathit{Hconv}(\overline{a}(\widetilde{w}, K_a, \emptyset, B').\mathbf{0}, |\widetilde{w}|, 1)$ is a MMCπ. By induction on the length of $\widetilde{w}$:
  - $|\widetilde{w}| = 0$, by *Hconv*'s definition, $\mathit{Hconv}(\overline{a}(K_a, \emptyset, B').\mathbf{0}, 0, 1) = \overline{a}(K_a, \emptyset, B').\mathbf{0}$. Since K_a is a variable, $\emptyset$ is modelled as an empty list (cf. Section 5), and there are only names in B', $\overline{a}(K_a, \emptyset, B').\mathbf{0}$ is a MMCπ.
  - We now assume $\mathit{Hconv}(\overline{a}(w_1, w_2, ..., w_n, K_a, \emptyset, B').\mathbf{0}, n, 1)$ is a MMCπ. Such a process looks as follows:

$$
\begin{array}{l}
\mathbf{new}\ h_j\ (\\
\quad \mathbf{new}\ h_{j-1}\ (\\
\qquad \cdots (\mathbf{new}\ h_1\ (\overline{a}(\widetilde{y}, K_a, \emptyset, B').\mathbf{0})) \cdots \\
\quad) \\
)
\end{array}
$$

    Assuming in $(w_1, ..., w_n)$ there are j higher order expressions, and being $\widetilde{y}$ the resulting output names after higher order expressions have been replaced. We have $|\widetilde{y}| = n$, and

$$
y_i = \begin{cases} w_i & \text{if } w_i \text{ is not higher-order} \\ h_m & \text{with } m \leq j \text{ and } (h_m, w_i) \in H \\ & \text{and } \nexists l : l \leq n, l \neq i, w_l = h_m, \text{ otherwise} \end{cases}
$$

    By induction, such a process is a MMCπ. We need to show for a new w_{n+1}, $\mathit{Hconv}(\overline{a}(w_1, w_2, ..., w_n, w_{n+1}, K_a, \emptyset, B').\mathbf{0}, n+1, 1)$ is a MMCπ as well. If w_{n+1} is a name (i.e., not a higher-order expression), then by the definition of *Hconv*, the resulting process is (changes with respect to the process for n are double-underlined):

$$
\begin{array}{l}
\mathbf{new}\ h_j\ (\\
\quad \mathbf{new}\ h_{j-1}\ (\\
\qquad \cdots (\mathbf{new}\ h_1\ (\overline{a}(\underline{\underline{\widetilde{y}'}}, k_a, \emptyset, B').\mathbf{0})) \cdots \\
\quad) \\
)
\end{array}
$$

    Notice the only difference between the resulting process, and the process for $\mathit{Hconv}(\overline{a}(w_1, w_2, ..., w_n, K_a, \emptyset, B').\mathbf{0}, n, 1)$, is the use of $\widetilde{y'}$ instead of $\widetilde{y}$, where $\widetilde{y'} = y_1, y_2, ..., y_n, w_{n+1}$. Since this change only increases the number of names written by one, the resulting process is a MMCπ. If

the new w_{n+1} is a higher-order expression, by the definition of *Hconv*, the resulting process is (changes are double-underlined):

$$\begin{array}{l}\underline{\underline{\textbf{new}\ h\ (}}\\ \quad \textbf{new}\ h_j\ (\\ \quad\quad \textbf{new}\ h_{j-1}\ (\\ \quad\quad\quad \cdots (\textbf{new}\ h_1\ (\overline{a}(\underline{\underline{\widetilde{y''}}}, K_a, \emptyset, B').\mathbf{0}))\cdots\\ \quad\quad)\\ \quad)\\ \underline{\underline{)}}\end{array}$$

Where $\widetilde{y''} = y_1, y_2, ..., y_n, h$, with h a fresh name used as a higher-order indicator and $H = H \cup \{(h, w_{n+1})\}$. Since the changes only introduce a restriction **new** h and a name h to the output list, the resulting expression is a MMCπ expression, and therefore, the following is a MMCπ:

$$\mathit{Hconv}(\overline{a}(w_1, w_2, ..., w_{n+1}, K_a, \emptyset, B').\mathbf{0}, n+1, 1)$$

We have shown $\mathit{Hconv}(\overline{a}(\widetilde{w}, K_a, \emptyset, B').\mathbf{0}, |\widetilde{w}|, 1)$, for $|\widetilde{w}| = 0$ and $|\widetilde{w}| = n+1$ are both a MMCπ; then by induction on the length of $\widetilde{w}$, $\mathit{Hconv}(\ \overline{a}(\widetilde{w}, K_a, \emptyset, B').\mathbf{0}, |\widetilde{w}|, 1)$ is a MMCπ for all $|\overline{a}(\widetilde{w})| = n$. And because $\mathit{Hconv}(\overline{a}(\widetilde{w}, K_a, \emptyset, B').\mathbf{0}, |\widetilde{w}|, 1)$ is a MMCπ, then $\mathcal{I}[\![\ B, \overline{a}(\widetilde{w})\ ]\!]$ is a MMCπ, therefore $\mathcal{I}[\![\ B, P\ ]\!]$ is a MMCπ. In general, $\mathit{Hconv}(\overline{a}(\widetilde{w}, K_a, \emptyset, B').R, |\widetilde{w}|, 1)$ is a MMCπ if R is a MMCπ. Therefore, $\mathit{Hwrite}(\overline{a}(\widetilde{w}, \emptyset), R, B\ \{a\})$ is a MMCπ when R is a MMCπ. We will make use of this observation when showing that $\mathcal{I}[\![\ B, K[Q]\ ]\!]$ is a MMCπ.

- $\overline{a}(\widetilde{w}, K_s)$ with K_s a kell containment set. The argument is similar to that of $\overline{a}(\widetilde{w})$, but with K_s instead of $\emptyset$. When $\mathit{Hconv}(\overline{a}(\widetilde{w}, K_a, K_s, B').\mathbf{0}, |\widetilde{w}|, 1)$ with K_a a variable, the resulting expression is MMCπ, because K_s is represented as a list. The only difference is K_s may not be empty, but in such a case K_s as part of a communication is a valid MMCπ.

- $a(\widetilde{c}) \triangleright Q$, by definition $\mathcal{I}[\![\ B, a(\widetilde{c}) \triangleright Q\ ]\!] \stackrel{def}{=} a(\widetilde{c}, \emptyset, K_c, \mathit{bnd}).\texttt{code}(\texttt{inst}(Q, \widetilde{c}, \mathit{bnd}, Q_\pi), Q_\pi)$ with K_c a variable. Because $\widetilde{c}$ is a sequence, possibly empty, of names and variables, $\emptyset$ is a valid value in MMCπ communications, and *bnd* is a variable, an expression $a(\widetilde{c}, \emptyset, K_c, \mathit{bnd}).P$ is a MMCπ, if P is a MMCπ.

  `code(Op, R)` is a MMC extension of the π-calculus. It is defined as performing the predicate `Op`, and then behaving as `R`. In our case, `Op` is $\texttt{inst}(Q, \widetilde{c}, \mathit{bnd}, Q_\pi)$, and `R` is Q_π. We need to show that after evaluating $\texttt{inst}(Q, \widetilde{c}, \mathit{bnd}, Q_\pi)$, Q_π is a MMCπ. By definition, $\texttt{inst}(Q, \widetilde{c}, \mathit{bnd}, Q_\pi)$ `:-` $Q_\pi = \mathcal{I}[\![\ B \cup \mathit{bnd}, \mathcal{F}[\![\ \mathcal{H}[\![\ Q\ ]\!]\ ]\!]\ ]\!]$. Recall $\mathcal{H}[\![\ Q\ ]\!]$ replaces higher-order indicators found in Q with their corresponding associated process expressions in H. The result of the instantiation is a kell-m process, which is then processed by $\mathcal{F}$, so that fresh names are used. The result of the fresh-names processing is again a kell-m process.

  We will show $\mathcal{H}[\![\ Q\ ]\!]$ is a kell-m process for any kell-m process Q. Similarly, $\mathcal{F}[\![\ Q\ ]\!]$ can be shown to be a kell-m process (we omit the proof). Based on Q and $\mathcal{H}$'s definition, the possible transformations made to Q are:

  - $Q = \mathbf{0}$, by definition $\mathcal{H}[\![\ \mathbf{0}\ ]\!] \stackrel{def}{=} \mathbf{0}$, and $\mathbf{0}$ is trivially a kell-m process.
  - $Q = x$, if x is a higher-order indicator, by definition $\mathcal{H}[\![\ x\ ]\!] \stackrel{def}{=} P'$, where P' is the kell-m process associated to x in H. Since P' is a kell-m process then $\mathcal{H}[\![\ h\ ]\!]$ is a kell-m process. If x is not a higher-order indicator, then x is a process variable, and process variables are kell-m processes.
  - $Q = d(\widetilde{b}) \triangleright R$, by definition $\mathcal{H}[\![\ d(\widetilde{b})\ ]\!] \triangleright R) \stackrel{def}{=} d(\widetilde{b}) \triangleright \mathcal{H}[\![\ R\ ]\!]$. By structural induction, assuming $\mathcal{H}[\![\ R\ ]\!]$ is a kell-m process, then $d(\widetilde{b}) \triangleright \mathcal{H}[\![\ R\ ]\!]$ is also a kell-m process.
  - $Q = K[x] \triangleright R$, by definition $\mathcal{H}[\![\ K[x] \triangleright R\ ]\!] \stackrel{def}{=} K[x] \triangleright \mathcal{H}[\![\ R\ ]\!]$. Assuming, by structural induction, if $\mathcal{H}[\![\ R\ ]\!]$ is a kell-m process, then $K[x] \triangleright \mathcal{H}[\![\ R\ ]\!]$ is also a kell-m process.
  - $Q = K[R]$, by definition $\mathcal{H}[\![\ K[R]\ ]\!] \stackrel{def}{=} K[\mathcal{H}[\![\ R\ ]\!]]$. Assuming, by structural induction, that $\mathcal{H}[\![\ R\ ]\!]$ is a kell-m process, then $K[\mathcal{H}[\![\ R\ ]\!]]$ is also a kell-m process.
  - $Q = \overline{d}(\widetilde{b})$, trivially a kell-m process, since by definition there is no modification of the kell-m process $\overline{d}(\widetilde{b})$: $\mathcal{H}[\![\ \overline{d}(\widetilde{b})\ ]\!] \stackrel{def}{=} \overline{d}(\widetilde{b})$.

– $Q = R(\widetilde{v})$, by definition $\mathcal{H}[\![\ R(\widetilde{v})\ ]\!] \stackrel{def}{=} \mathcal{H}[\![\ R_d\{\widetilde{v}/\widetilde{y}\}\ ]\!]$ when $R(\widetilde{y}) = R_d$. Since R_d is a kell-m process, by structural induction $\mathcal{H}[\![\ R_d\{\widetilde{v}/\widetilde{y}\}\ ]\!]$ is also a kell-m process.

– $Q = R \mid T$, by definition $\mathcal{H}[\![\ R|T\ ]\!] \stackrel{def}{=} \mathcal{H}[\![\ R\ ]\!] \mid \mathcal{H}[\![\ T\ ]\!]$. By structural induction, assuming $\mathcal{H}[\![\ R\ ]\!]$ and $\mathcal{H}[\![\ T\ ]\!]$ are kell-m processes, then their parallel composition is also a kell-m process.

Because $\mathcal{F}[\![\ \mathcal{H}[\![\ Q\ ]\!]\ ]\!]$ is a kell-m process, by structural induction, $\mathcal{I}[\![\ B \cup \mathit{bnd}, \mathcal{F}[\![\ \mathcal{H}[\![\ Q\ ]\!]\ ]\!]\ ]\!]$ is a MMCπ if:

- $a(\widetilde{c}, K_s) \triangleright Q$, with K_s a kell containment set, possibly empty. The argument is the same as for $a(\widetilde{c}) \triangleright Q$ above. Notice $a(\widetilde{c}, K_s, K_c, \mathit{bnd}).P$ is a MMCπ if P is a MMCπ.

- $K[Q]$, by $\mathcal{I}$'s definition $\mathcal{I}[\![\ B, K[Q]\ ]\!] \stackrel{def}{=} \mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]$ if Q is not the parallel composition of processes R and T; otherwise,

$$\mathcal{I}[\![\ B, K[Q]\ ]\!] \stackrel{def}{=} \mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[R|T]\ ]\!]\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|R]\ ]\!]\ ]\!]$$

First we will show when $Q = \mathbf{0}$, then $\mathcal{I}[\![\ B, K[Q]\ ]\!]$ is a MMCπ. By definition, $\mathcal{I}[\![\ B, K[\mathbf{0}]\ ]\!] \stackrel{def}{=} \mathcal{S}[\![\ \mathbf{0}, B, K[\mathbf{0}], \emptyset\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[\mathbf{0}]\ ]\!]\ ]\!]$. By $\mathcal{S}$'s definition:

$$\begin{array}{rcl} \mathcal{S}[\![\ \mathbf{0}, B, K[\mathbf{0}], \emptyset\ ]\!] & \stackrel{def}{=} & \mathit{Hwrite}(\overline{K}(\mathbf{0}, \emptyset), \mathcal{I}[\![\ B, \mathbf{0}\{\mathbf{0}/K[\mathbf{0}]\}\ ]\!], B \setminus \{K\}) \\ & & \quad + \ \mathcal{S}[\![\ \mathrm{K[\mathbf{0}]}, B, \mathbf{0}, \{K\}\ ]\!] \\ & \equiv & \mathit{Hwrite}(\overline{K}(\mathbf{0}, \emptyset), \mathcal{I}[\![\ B, \mathbf{0}\ ]\!], B \setminus \{K\}) \ + \ \mathbf{0} \\ & \equiv & \mathit{Hwrite}(\overline{K}(\mathbf{0}, \emptyset), \mathbf{0}, B \setminus \{K\}) \ + \ \mathbf{0} \end{array}$$

By $\mathcal{A}$'s definition, $\mathcal{I}[\![\ B, \mathcal{A}[\![\ K[\mathbf{0}]\ ]\!]\ ]\!] \equiv \mathcal{I}[\![\ B, \mathbf{0}\ ]\!] \equiv \mathbf{0}$. Therefore:

$$\mathcal{I}[\![\ B, K[\mathbf{0}]\ ]\!] \equiv \mathit{Hwrite}(\overline{K}(\mathbf{0}, \emptyset), \mathbf{0}, B \setminus \{K\}) \ + \ \mathbf{0} \ + \ \mathbf{0}$$

We have already shown $\mathit{Hwrite}(\overline{K}(\mathbf{0}, \emptyset), \mathbf{0}, B \setminus \{K\})$ is a MMCπ. Finally, the choice of $\mathit{Hwrite}(\overline{K}(\mathbf{0}, \emptyset), \mathbf{0}, B \setminus \{K\})$, $\mathbf{0}$, and $\mathbf{0}$ is trivially a MMCπ.

For the general case, we need to show $\mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!]$ is a MMCπ, then we will show $\mathcal{A}[\![\ K[Q]\ ]\!]$ produces a kell-m process. We will argue, by structural induction, that $\mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]$ is therefore a MMCπ.

By $\mathcal{S}$'s definition, $\mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!] \stackrel{def}{=} \mathit{Hwrite}(\overline{K}(Q, \emptyset), \mathcal{I}[\![\ B, \mathbf{0}\{\mathbf{0}/K[Q]\}\ ]\!], B \setminus \{K\}) + \mathcal{S}[\![\ \mathrm{K[Q]}, B, Q, \{K\}\ ]\!]$. Since $\mathcal{I}[\![\ B, \mathbf{0}\{\mathbf{0}/K[Q]\}\ ]\!] \equiv \mathcal{I}[\![\ B, \mathbf{0}\ ]\!] \stackrel{def}{=} \mathbf{0}$, then
$\mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!] \equiv \mathit{Hwrite}(\overline{K}(Q, \emptyset), \mathbf{0}, B \setminus \{K\}) \ + \ \mathcal{S}[\![\ \mathrm{K[Q]}, B, Q, \{K\}\ ]\!]$. Because $\mathit{Hwrite}(\overline{K}(Q, \emptyset), \mathbf{0}, B \setminus \{K\})$ is a MMCπ, we just need to show:

$$\mathcal{S}[\![\ \mathrm{K[Q]}, B, Q, \{K\}\ ]\!] \text{ is a MMC}\pi.$$

The possible cases for Q are:

– Q is not a kell $G[R]$ or a parallel composition of two processes $R|T$, then by $\mathcal{S}$'s definition, $\mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] \stackrel{def}{=} \mathbf{0}$, and $\mathbf{0}$ is trivially a MMCπ.

– $G[R]$, by $\mathcal{S}$'s definition:
$\mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] \stackrel{def}{=} \mathit{Hwrite}(\overline{G}(R, \{K\}), \mathcal{I}[\![\ B, K[\mathbf{0}]\ ]\!], B \setminus \{G\}) + \mathcal{S}[\![\ K[Q], B, R, \{K, G\}\ ]\!]$
We have already shown $\mathcal{I}[\![\ B, K[\mathbf{0}]\ ]\!]$ is a MMCπ therefore, $\mathit{Hwrite}(\overline{G}(R, \{K\}), \mathcal{I}[\![\ B, K[\mathbf{0}]\ ]\!], B \setminus \{G\})$ is a MMCπ (recall, $\mathit{Hwrite}(\overline{a}(\widetilde{w}, K_s), T, B)$ is a MMCπ if T is a MMCπ). Still to show is that $\mathcal{S}[\![\ K[Q], B, R, \{K, G\}\ ]\!]$ is a MMCπ. Interesting cases occur when R is the nesting of multiple kells, possibly composed in parallel. For example $K_1[K_2[...K_n[T]...] \mid K_u[W]]$, where T and W are not parallel composition of processes or another kell. In such cases, the result of $\mathcal{S}$ looks as:

$$\begin{array}{ll} \textbf{new}\ h_u\ \overline{K_u}(h_u, \{K, G, K_1\}).\mathcal{I}[\![\ B, K[G[K_1[K_2[...K_n[T]...] \mid \mathbf{0}]]]\ ]\!] \\ + \quad \textbf{new}\ h_1\ \overline{K_1}(h_1, \{K, G\}).\mathcal{I}[\![\ B, K[G[\mathbf{0}]]\ ]\!] \\ + \quad \textbf{new}\ h_2\ \overline{K_2}(h_2, \{K, G, K_1\}).\mathcal{I}[\![\ B, K[G[K_1[\mathbf{0} \mid K_u[W]]]]\ ]\!] \\ + \quad \cdots \\ + \quad \textbf{new}\ h_n\ \overline{K_n}(h_n, \{K, G, K_1, ..., K_{n-1}\}). \\ \quad \mathcal{I}[\![\ B, K[G[K_1[K_2[...\mathbf{0}...] \mid K_u[W]]]]\ ]\!] \end{array}$$

With

$$H = H \cup \{ \quad (h_u, W), \\ (h_1, K_2[K_3[...K_n[T]...]] \mid K_u[W]), \\ (h_2, K_3[K_4[...K_n[T]...]] \mid K_u[W]), \\ ..., \\ (h_n, T)\}$$

Names hs are higher-order indicators. The channel outputs $\overline{K_i}(h_i, K_s)$ are MMCπ expressions. For the expressions,

$$\mathcal{I}[\![\ B, K[G[K_1[K_2[...]]]]\ ]\!]$$

$\mathcal{I}$'s definition uses the advance function $\mathcal{A}$:

$$\mathcal{I}[\![\ B, K[G[K_1[K_2[...]]]]\ ]\!] \stackrel{def}{=} \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[G[K_1[K_2[...]]]]\ ]\!]\ ]\!]$$

Assuming $\mathcal{A}$ produces a kell-m process (we will show this later), by structural induction, the encoding $\mathcal{I}$ of such process is a MMCπ. Therefore, $\mathcal{S}[\![\ K[G[R]],\ B,\ G[R],\ \{K\}\ ]\!]$, is the composition, via the π-calculus choice operator $+$, of MMCπs. The choice of MMCπs is trivially a MMCπ.

– $R|T$, by $\mathcal{S}$'s definition, $\mathcal{S}[\![\ K[Q], B, R \mid T\ ]\!] \stackrel{def}{=} \mathcal{S}[\![\ K[Q], B, R, \{K\}\ ]\!] +$
$\mathcal{S}[\![\ K[Q], B, T, \{K\}\ ]\!]$. By structural induction on $\mathcal{S}$, both $\mathcal{S}[\![\ K[Q], B, R, \{K\}\ ]\!]$ and $\mathcal{S}[\![\ K[Q], B, T, \{K\}\ ]\!]$ are a MMCπ, and the choice of two MMCπs is trivially a MMCπ.

We will now show the result of $\mathcal{A}[\![\ K[Q]\ ]\!]$ is a kell-m process. By structural induction on $\mathcal{A}$, the cases for Q are:

– $\mathbf{0}$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[\mathbf{0}]\ ]\!] \stackrel{def}{=} \mathbf{0}$, and $\mathbf{0}$ is a kell-m process.

– **new** $c\ R$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[\mathbf{new}\ c\ R]\ ]\!] \stackrel{def}{=} \mathbf{new}\ c\ \mathcal{A}[\![\ K[R]\ ]\!]$. By structural induction, $\mathcal{A}[\![\ K[R]\ ]\!]$ is a kell-m process, and the restriction of a kell-m process is a kell-m process.

– $\overline{a}(\widetilde{w})$ by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[\overline{a}(\widetilde{w})]\ ]\!] \stackrel{def}{=} \overline{a}(\widetilde{w}, \{K\})$, and $\overline{a}(\widetilde{w}, \{K\})$ is a kell-m process.

– $\overline{a}(\widetilde{w}, K_s)$ by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[\overline{a}(\widetilde{w}, K_s)]\ ]\!] \stackrel{def}{=} \overline{a}(\widetilde{w}, K'_s)$, with $K' = K_s \cup \{K\}$, and $\overline{a}(\widetilde{w}, K'_s)$ is a kell-m process.

– $a(\widetilde{c}) \triangleright R$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[a(\widetilde{c}) \triangleright R]\ ]\!] \stackrel{def}{=} a(\widetilde{c}, \{K\}) \triangleright K[R]$. This is a kell-m channel trigger expression.

– $a(\widetilde{c}, K_s) \triangleright R$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[a(\widetilde{c}, K_s) \triangleright R]\ ]\!] \stackrel{def}{=} a(\widetilde{c}, K'_s) \triangleright K[R]$ with $K'_s = K_s \cup \{K\}$. This is a kell-m channel trigger expression.

– $L[x] \triangleright R$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[L[x] \triangleright R]\ ]\!] \stackrel{def}{=} L(x, \{K\}) \triangleright K[R]$. This is a kell-m channel trigger expression.

– $L[R]$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[L[R]]\ ]\!] \stackrel{def}{=} \mathcal{A}[\![\ K[\mathcal{A}[\![\ L[R]\ ]\!]]\ ]\!]$. By structural induction, $\mathcal{A}[\![\ L[R]\ ]\!]$ is a kell-m process. $\mathcal{A}$ eliminates kells with null processes, and lifts name restrictions (**new** c), channel concretions ($\overline{a}(\widetilde{a})$) and abstractions ($a(\widetilde{c}) \triangleright U$, $K[x] \triangleright U$) from nested kells outwards. The invocation $\mathcal{A}[\![\ K[\mathcal{A}[\![\ L[R]\ ]\!]]\ ]\!]$ just lifts to the outside of K whatever expression was, in turn, lifted from $L[R]$. The expression being lifted has the form $\mathbf{0}$, $\mathcal{A}[\![\ K[\mathbf{0}]\ ]\!]$, **new** $c\ R$, $a(\widetilde{c}) \triangleright R$, or $L[x] \triangleright R$. We have already shown that the result of advancing on such expressions is also a kell-m expression.

– $R(\widetilde{w})$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[R(\widetilde{w})]\ ]\!] \stackrel{def}{=} \mathcal{A}[\![\ K[R_d\{\widetilde{w}/\widetilde{y}\}]\ ]\!]$ where $R(\widetilde{y}) \stackrel{def}{=} R_d$. By structural induction $\mathcal{A}[\![\ K[R_d\{\widetilde{w}/\widetilde{y}\}]\ ]\!]$ is a kell-m process.

– $R|T$, R can be one of:

  * $\mathbf{0}$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[\mathbf{0} \mid T]\ ]\!] \stackrel{def}{=} \mathcal{A}[\![\ K[T]\ ]\!]$, and by structural induction, $\mathcal{A}[\![\ kkKT\ ]\!]$ is a kell-m process.

  * **new** $c\ U$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[\mathbf{new}\ c\ U|T]\ ]\!] \stackrel{def}{=} \mathbf{new}\ c\ K[U|T]$. A restriction of a kell-m process is a kell-m process.

  * $\overline{a}(\widetilde{w})$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[\overline{a}(\widetilde{w}) \mid T]\ ]\!] \stackrel{def}{=} \overline{a}(\widetilde{w}, \{K\}) \mid K[T]$. Parallel composition of kell-m processes is a kell-m process.

  * $\overline{a}(\widetilde{w}, K_s)$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[\overline{a}(\widetilde{w}, K_s) \mid T]\ ]\!] \stackrel{def}{=} \overline{a}(\widetilde{w}, K_s \cup \{K\}) \mid K[T]$. Parallel composition of kell-m processes is a kell-m process.

α	α'	
τ	$\tau(\mathit{concr}, K_a, K_c)$	with *concr* concretion, K_a, K_c kell cont. sets
$\overline{a}(\widetilde{w})$	$\overline{a}(\widetilde{w}', K_a, K_c, B)$	with K_a variable, K_c kell cont. set
$a(\widetilde{c})$	$a(\widetilde{c}, K_a, K_c, \mathit{bnd})$	with K_a kell cont. set, K_c variable
$\overline{K}[P]$	$\overline{K}(\textbf{new}\ h_P\ , K_a, K_c, B)$	with K_a variable, K_c kell cont. set
$K[x]$	$K(x, K_a, K_c, \mathit{bnd})$	with K_a kell cont. set, K_c variable

Table 1: Kell-m and Corresponding MMCπ Actions

- $a(\widetilde{c}) \triangleright U$, by $\mathcal{A}$'s definition, $\mathcal{A}[\![\ K[(a(\widetilde{c}) \triangleright U) \mid T]\ ]\!] \stackrel{def}{=} a(\widetilde{c}, \{K\}) \triangleright K[U \mid T]$. Trigger expressions that match channel outputs are kell-m processes.
- $a(\widetilde{c}, K_s) \triangleright U$, by $\mathcal{A}$'s definition, $\mathcal{A}[\![\ K[(a(\widetilde{c}, K_s) \triangleright U) \mid T]\ ]\!] \stackrel{def}{=} a(\widetilde{c}, K_s \cup \{K\}) \triangleright K[U \mid T]$. Trigger expressions that match channel outputs are kell-m processes.
- $L[x] \triangleright U$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[(L[x] \triangleright U) \mid T]\ ]\!] \stackrel{def}{=} L(x, \{K\}) \triangleright K[U \mid T]$. Trigger expressions that match kells are kell-m processes.
- $H[U]$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[H[U] \mid T]\ ]\!] \stackrel{def}{=} \mathcal{A}[\![\ K[\mathcal{A}[\![\ H[U]\ ]\!] \mid T]\ ]\!]$. By structural induction $\mathcal{A}[\![\ H[U]\ ]\!]$ is a kell-m process. $\mathcal{A}[\![\ H[U]\ ]\!]$ has the form $\mathbf{0}$, $\mathcal{A}[\![\ K[\mathbf{0}]\ ]\!]$, **new** $c\ R$, $a(\widetilde{c}) \triangleright R$, or $L[x] \triangleright R$. We have already shown that advancing the execution of such an expression, composed in parallel with another kell-m expression, is also a kell-m expression. Hence, the result of $\mathcal{A}[\![\ K[H[U] \mid T]\ ]\!]$ is a kell-m process.
- $U(\widetilde{w})$, by $\mathcal{A}$'s definition $\mathcal{A}[\![\ K[U(\widetilde{w}) \mid T]\ ]\!] \stackrel{def}{=} \mathcal{A}[\![\ K[U_d\{\widetilde{w}/\widetilde{y}\} \mid T]\ ]\!]$ where $U(\widetilde{y}) \stackrel{def}{=} U_d$. By structural induction $\mathcal{A}[\![\ K[U_d\{\widetilde{w}/\widetilde{y}\} \mid T]\ ]\!]$ is a kell-m process.

Without loss of generality, $\mathcal{A}[\![\ K[T|R]\ ]\!]$ is also a kell-m process.

Since both $\mathcal{S}[\![\ K[Q], B, K[Q], \{K\}\ ]\!]$ and $\mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]$ are a MMCπ, then their composition via the π-calculus choice operator is also a MMCπ. Therefore $\mathcal{I}[\![\ B, P\ ]\!]$ is a MMCπ. In the case of Q being the parallel composition of processes R and T, we have also shown $\mathcal{I}[\![\ B, \mathcal{A}[\![\ K[R|T]\ ]\!]\ ]\!]$ and $\mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|R]\ ]\!]\ ]\!]$ are MMCπ, therefore, $\mathcal{I}[\![\ B, K[Q]\ ]\!]$ is a MMCπ.

- $K[x] \triangleright Q$, by $\mathcal{I}$'s definition $\mathcal{I}[\![\ B, K[x] \triangleright Q\ ]\!] \stackrel{def}{=} \mathcal{I}[\![\ B, \mathcal{F}[\![\ K(x, \emptyset) \triangleright Q\ ]\!]\ ]\!]$. Because $\mathcal{F}$ only performs alpha-conversions, the kell-m process resulting of $\mathcal{F}[\![\ K(x, \emptyset) \triangleright Q\ ]\!]$ will have the form $K(x', \emptyset) \triangleright Q'$, and we have already shown that the result of $\mathcal{I}[\![\ B, K(x', \emptyset) \triangleright Q'\ ]\!]$ is a MMCπ, therefore $\mathcal{I}[\![\ B, P\ ]\!]$ is a MMCπ.
- $Q(\widetilde{w})$, by $\mathcal{I}$'s definition $\mathcal{I}[\![\ B, Q(\widetilde{w})\ ]\!] \stackrel{def}{=} \mathcal{I}[\![\ B, \mathcal{F}[\![\ Q_d\{\widetilde{w}/\widetilde{y}\}\ ]\!]\ ]\!]$ where $Q(\widetilde{y}) \stackrel{def}{=} Q_d$. Because, $\mathcal{F}$ only performs alpha-conversions, by structural induction, $\mathcal{I}[\![\ B, \mathcal{F}[\![\ Q_d\{\widetilde{w}/\widetilde{y}\}\ ]\!]\ ]\!]$ is a MMCπ, therefore $\mathcal{I}[\![\ P\ ]\!]$ is a MMCπ.
- $Q \mid R$, by definition $\mathcal{I}[\![\ B, Q|R\ ]\!] \stackrel{def}{=} \mathcal{I}[\![\ B, Q\ ]\!] \mid \mathcal{I}[\![\ B, R\ ]\!]$. By induction both $\mathcal{I}[\![\ Q\ ]\!]$ and $\mathcal{I}[\![\ R\ ]\!]$ are MMCπs, and because the parallel composition of MMCπs is a MMCπ, then $\mathcal{I}[\![\ P\ ]\!]$ is a MMCπ.

We have shown that the result of the interpretation of a kell-m process is a MMC π-calculus process.

5.7 P is Indistinguishable from $\mathcal{I}[\![\ \{\}, P\ ]\!]$

We want to show that when a kell-m process P transitions to Q, its encoding $\mathcal{I}[\![\ B, P\ ]\!]$, transitions to Q's encoding: if $P \xrightarrow{\alpha} Q$, then $\mathcal{I}[\![\ B, P\ ]\!] \xrightarrow{\alpha'} \mathcal{I}[\![\ B', Q\ ]\!]$.

As shown in Table 1, the names involved in the actions α and α' are the same, but the actual parameters of the actions may differ.

$\widetilde{w}'$ is $\widetilde{w}$ with higher-order expressions replaced by higher-order indicators, and h_P is the higher-order indicator associated with P. The sets and variables used to expose kell containment information and bound names are included in the MMCπ actions. Because the names of the channels in communications do not change, an observer will not notice a change in the visibility predicates between P and its encoding, nor between Q and its encoding.

We start by defining in Figure 23 $\mathcal{V}_M$, the visibility predicates for MMC. We do not specify a visibility predicate

$$
\begin{array}{lcl}
\mathcal{V}_M[\![\ \mathbf{0}\ ]\!] & \stackrel{def}{=} & \{\} \\
\mathcal{V}_M[\![\ \textbf{new}\ a\ P\ ]\!] & \stackrel{def}{=} & \mathcal{V}_M[\![\ P\ ]\!] \setminus \{a\} \\
\mathcal{V}_M[\![\ \overline{a}(\widetilde{w}).P\ ]\!] & \stackrel{def}{=} & \{\overline{a}\} \\
\mathcal{V}_M[\![\ a(\widetilde{c}).P\ ]\!] & \stackrel{def}{=} & \{a\} \\
\mathcal{V}_M[\![\ P|Q\ ]\!] & \stackrel{def}{=} & \mathcal{V}_M[\![\ P\ ]\!] \cup \mathcal{V}_M[\![\ Q\ ]\!] \\
\mathcal{V}_M[\![\ P+Q\ ]\!] & \stackrel{def}{=} & \mathcal{V}_M[\![\ P\ ]\!] \cup \mathcal{V}_M[\![\ Q\ ]\!] \\
\mathcal{V}_M[\![\ \texttt{code(Op, P)}\ ]\!] & \stackrel{def}{=} & \mathcal{V}_M[\![\ P\ ]\!]
\end{array}
$$

Figure 23: Visibility Predicates for MMCπ

for $[a=b]P$ since such expressions are not generated by our encoding of kell-m processes. $\mathcal{V}$ the visibility for kell-m is defined in Figure 8, Section 3.3.

Using structural induction we will show the visibility predicates for P and its MMCπ encoding $\mathcal{I}[\![\ \{\}, P\ ]\!]$ are the same. For $\mathbf{0}$ and x the visibility predicates are trivially the same: both empty for P and $\mathcal{I}[\![\ \{\}, P\ ]\!]$. The other cases are:

- **new** $a\ Q$, by the definition of $\mathcal{V}$, $\mathcal{V}_M$ and $\mathcal{I}$ we have $\mathcal{V}[\![\ \textbf{new}\ a\ Q\ ]\!] \stackrel{def}{=} \mathcal{V}[\![\ Q\ ]\!] \setminus \{a\}$, $\mathcal{V}_M[\![\ \textbf{new}\ a\ \mathcal{I}[\![\ B \cup \{a\}, Q\ ]\!]\ ]\!] \stackrel{def}{=} \mathcal{V}[\![\ \mathcal{I}[\![\ B \cup \{a\}, Q\ ]\!]\ ]\!] \setminus \{a\}$. By structural induction, $\mathcal{V}[\![\ Q\ ]\!] = \mathcal{V}[\![\ \mathcal{I}[\![\ B \cup \{a\}, Q\ ]\!]\ ]\!]$, therefore, $\mathcal{V}[\![\ \textbf{new}\ a\ Q\ ]\!] = \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \textbf{new}\ a\ Q\ ]\!]\ ]\!]$.

- $\overline{a}(\widetilde{w})$, by $\mathcal{I}$'s definition, $\mathcal{I}[\![\ B, \overline{a}(\widetilde{w})\ ]\!] \stackrel{def}{=} \mathit{Hwrite}(\overline{a}(\widetilde{w}, \emptyset), \mathbf{0}, B \setminus \{a\})$. Recall *Hwrite* replaces higher-order expressions in $\widetilde{w}$ with higher-order indicator names, and keeps track of those names in H. By $\mathcal{V}_M$'s and *Hwrite*'s definition: $\mathcal{V}_M[\![\ \mathit{Hwrite}(\overline{a}(\widetilde{w}, \emptyset), \mathbf{0}, B \setminus \{a\})\ ]\!] \stackrel{def}{=} \{\overline{a}\}$. Since, $\mathcal{V}[\![\ \overline{a}(\widetilde{w})\ ]\!] \stackrel{def}{=} \{\overline{a}\}$, then $\mathcal{V}[\![\ \overline{a}(\widetilde{w})\ ]\!] = \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \overline{a}(\widetilde{w})\ ]\!]\ ]\!]$.

- $a(\widetilde{c}) \triangleright Q$, by $\mathcal{V}$'s definition $\mathcal{V}[\![\ a(\widetilde{c}) \triangleright Q\ ]\!] \stackrel{def}{=} \{a\}$. By $\mathcal{V}_M$'s and $\mathcal{I}$'s definition:

$$
\begin{array}{c}
\mathcal{I}[\![\ B, a(\widetilde{c}) \triangleright Q\ ]\!] \stackrel{def}{=} a(\widetilde{c}, K_s, K_c, \mathit{bnd}).\texttt{code}(\texttt{inst}(Q, \widetilde{c}, \mathit{bnd}, Q_\pi), Q_\pi) \\
\mathcal{V}[\![\ a(\widetilde{c}, \mathit{bnd}).\texttt{code}(\texttt{inst}(Q, \widetilde{c}, \mathit{bnd}, Q_\pi), Q_\pi)\ ]\!] \stackrel{def}{=} \{a\}
\end{array}
$$

  Then $\mathcal{V}[\![\ a(\widetilde{c}) \triangleright Q\ ]\!] = \mathcal{V}_M[\![\ a(\widetilde{c}, \mathit{bnd}).\texttt{code}(\texttt{inst}(Q, \widetilde{c}, \mathit{bnd}, Q_\pi), Q_\pi)\ ]\!]$

- $K[x] \triangleright Q$, by $\mathcal{V}$'s definition $\mathcal{V}[\![\ K[x] \triangleright Q\ ]\!] \stackrel{def}{=} \{K\}$. By $\mathcal{I}$'s and $\mathcal{V}_M$'s definitions:

$$
\begin{array}{l}
\mathcal{V}_M[\![\ \mathcal{I}[\![\ B, K[x] \triangleright Q\ ]\!]\ ]\!] \\
\quad = \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{F}[\![\ K(x) \triangleright P\ ]\!]\ ]\!]\ ]\!] \\
\quad = \mathcal{V}_M[\![\ K(x,\mathrm{bnd}).\texttt{code}(\texttt{inst}(Q, x, \mathit{bnd}, Q_\pi), Q_\pi)\ ]\!] \\
\quad = \{K\} \\
\quad = \mathcal{V}[\![\ K[x] \triangleright Q\ ]\!]
\end{array}
$$

- $Q(\widetilde{w})$, by $\mathcal{V}$'s definition $\mathcal{V}[\![\ Q(\widetilde{w})\ ]\!] \stackrel{def}{=} \mathcal{V}[\![\ Q_d\{\widetilde{w}/\widetilde{y}\}\ ]\!]$, where $Q(\widetilde{y}) \stackrel{def}{=} Q_d$. By structural induction, $\mathcal{V}[\![\ Q_d\{\widetilde{w}/\widetilde{y}\}\ ]\!] = \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{F}[\![\ Q_d\{\widetilde{w}/\widetilde{y}\}\ ]\!]\ ]\!]\ ]\!]$.

- $K[Q]$, by $\mathcal{V}$'s definition, $\mathcal{V}[\![\ K[Q]\ ]\!] \stackrel{def}{=} \{\overline{K}\} \cup \mathcal{V}[\![\ Q\ ]\!]$. By $\mathcal{I}$'s definition $\mathcal{I}[\![\ B, K[Q]\ ]\!] \stackrel{def}{=} \mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]$ if Q is not the parallel composition of processes R and T; otherwise,$\mathcal{I}[\![\ B, K[Q]\ ]\!] \stackrel{def}{=} \mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[R|T]\ ]\!]\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|R]\ ]\!]\ ]\!]$. We need to show that when Q is not $R|T$:

$$
\begin{array}{l}
\mathcal{V}_M[\![\ \mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!]\ +\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]\ ]\!] \stackrel{def}{=} \\
\mathcal{V}_M[\![\ \mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!]\ ]\!] \cup \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]\ ]\!] = \\
\{\overline{K}\} \cup \mathcal{V}[\![\ Q\ ]\!]
\end{array}
$$

  If $Q \equiv R|T$, we need to show:

$$
\begin{array}{l}
\mathcal{V}_M[\![\ \mathcal{S}[\![\ \mathbf{0}, B, K[R|T], \emptyset\ ]\!]\ +\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[R|T]\ ]\!]\ ]\!]\ +\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|R]\ ]\!]\ ]\!]\ ]\!] \stackrel{def}{=} \\
\mathcal{V}_M[\![\ \mathcal{S}[\![\ \mathbf{0}, B, K[R|T], \emptyset\ ]\!]\ ]\!] \cup\ \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[R|T]\ ]\!]\ ]\!]\ ]\!] \cup \\
\quad \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|R]\ ]\!]\ ]\!]\ ]\!]\ =\ \{\overline{K}\} \cup \mathcal{V}[\![\ T|R\ ]\!]
\end{array}
$$

Notice:

$$\begin{array}{l}\mathcal{V}_M[\![\, \mathcal{S}[\![\, \mathbf{0}, B, K[Q], \emptyset \,]\!] \,]\!] \stackrel{def}{=} \\ \mathcal{V}_M[\![\, \mathit{Hwrite}(\overline{K}(Q,\emptyset), \mathcal{I}[\![\, B, \mathbf{0}\{\mathbf{0}/K[Q]\} \,]\!], B \setminus \{K\}) \;+\; \mathcal{S}[\![\, K[Q], B, Q, \{K\} \,]\!] \,]\!] = \\ \mathcal{V}_M[\![\, \mathit{Hwrite}(\overline{K}(Q,\emptyset), \mathcal{I}[\![\, B, \mathbf{0} \,]\!], B \setminus \{K\}) \,]\!] \cup \mathcal{V}_M[\![\, \mathcal{S}[\![\, K[Q], B, Q, \{K\} \,]\!] \,]\!] = \\ \mathcal{V}_M[\![\, \mathit{Hwrite}(\overline{K}(Q,\emptyset), \mathbf{0}, B \setminus \{K\}) \,]\!] \cup \mathcal{V}_M[\![\, \mathcal{S}[\![\, K[Q], B, Q, \{K\} \,]\!] \,]\!] = \\ \{\overline{K}\} \cup \mathcal{V}_M[\![\, \mathcal{S}[\![\, K[Q], B, Q, \{K\} \,]\!] \,]\!]\end{array}$$

Assuming Q is not $R|T$, left to show, is then:

$$\{\overline{K}\} \cup \mathcal{V}_M[\![\, \mathcal{S}[\![\, K[Q], B, Q, \{K\} \,]\!] \,]\!] \cup \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q] \,]\!] \,]\!] \,]\!] = \{\overline{K}\} \cup \mathcal{V}[\![\, Q \,]\!]$$

If Q is not a kell, then, by $\mathcal{S}$'s definition, $\mathcal{V}_M[\![\, \mathcal{S}[\![\, K[Q], B, Q, \{K\} \,]\!] \,]\!] \stackrel{def}{=} \mathcal{V}_M[\![\, \mathbf{0} \,]\!] \stackrel{def}{=} \{\}$. So, for such a Q, we need only to show: $\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q] \,]\!] \,]\!] \,]\!] = \mathcal{V}[\![\, Q \,]\!]$. The possible cases for Q are:

- $\mathbf{0}$, then, $\mathcal{A}[\![\, K[Q] \,]\!] = \mathbf{0}$, and therefore, $\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q] \,]\!] \,]\!] \,]\!] = \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathbf{0} \,]\!] \,]\!] = \{\} = \mathcal{V}[\![\, Q \,]\!]$.
- $\overline{a}(\widetilde{w})$, then, $\mathcal{A}[\![\, K[Q] \,]\!] = \overline{a}(\widetilde{w}, \{K\})$, and

$$\begin{array}{rcl}\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q] \,]\!] \,]\!] \,]\!] = & \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \overline{a}(\widetilde{w}, \{K\}) \,]\!] \,]\!] \\ = & \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \overline{a}(\widetilde{w}, \{K\}) \,]\!] \,]\!] \\ = & \mathcal{V}_M[\![\, \mathit{Hwrite}(\overline{a}(\widetilde{w}, \{K\}), \mathbf{0}, B \setminus \{a\}) \,]\!] \\ = & \{\overline{a}\} \\ = & \mathcal{V}[\![\, \overline{a}(\widetilde{w}) \,]\!]\end{array}$$

- $a(\widetilde{c}) \triangleright R$, then $\mathcal{A}[\![\, K[Q] \,]\!] = a(\widetilde{c}, \{K\}) \triangleright K[R]$, and

$$\begin{array}{rcl}\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q] \,]\!] \,]\!] \,]\!] & = & \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, a(\widetilde{c}, \{K\}) \triangleright K[R] \,]\!] \,]\!] \\ & = & \mathcal{V}_M[\![\, a(\widetilde{c}, \{K\}, K_c, \mathit{bnd}). \\ & & \qquad \texttt{code(inst}(R, \widetilde{c}, \mathit{bnd}, R_\pi), R_\pi) \,]\!] \\ & = & \{a\} \\ & = & \mathcal{V}[\![\, a(\widetilde{c}) \triangleright R \,]\!]\end{array}$$

- $L[x] \triangleright R$, then $\mathcal{A}[\![\, K[Q] \,]\!] = L(x, \{K\}) \triangleright K[R]$, and

$$\begin{array}{rcl}\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q] \,]\!] \,]\!] \,]\!] & = & \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, L(x, \{K\}) \triangleright K[R] \,]\!] \,]\!] \\ & = & \mathcal{V}_M[\![\, L(x, \{K\}, K_c, \mathit{bnd}). \\ & & \qquad \texttt{code(inst}(R, x, \mathit{bnd}, R_\pi), R_\pi) \,]\!] \\ & = & \{L\} \\ & = & \mathcal{V}[\![\, L[x] \triangleright R \,]\!]\end{array}$$

- $R(\widetilde{w})$, and $\mathcal{A}[\![\, R(\widetilde{w}) \,]\!] \stackrel{def}{=} \mathcal{A}[\![\, R_d(\{\widetilde{w}/\widetilde{y}\}) \,]\!]$, where $R(\widetilde{y}) \stackrel{def}{=} R_d$. By structural induction, we assume that:

$$\begin{array}{l}\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, R(\widetilde{w}) \,]\!] \,]\!] \,]\!] \\ \quad = \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, R_d(\{\widetilde{w}/\widetilde{y}\}) \,]\!] \,]\!] \,]\!] \\ \quad = \mathcal{V}[\![\, R_d(\{\widetilde{w}/\widetilde{y}\}) \,]\!]\end{array}$$

- **new** $a\ R$, then $\mathcal{A}[\![\, K[\textbf{new a } R] \,]\!] = \textbf{new}\ a\ \mathcal{A}[\![\, K[R] \,]\!]$; assuming that R is not of the form **new** c ..., we have:

$$\begin{array}{rcl}\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[\textbf{new a } R] \,]\!] \,]\!] \,]\!] = & \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \textbf{new}\ a\ \mathcal{A}[\![\, K[R] \,]\!] \,]\!] \,]\!] \\ = & \mathcal{V}_M[\![\, \textbf{new}\ a\ \mathcal{I}[\![\, B \cup \{a\}, \mathcal{A}[\![\, K[R] \,]\!] \,]\!] \,]\!]\end{array}$$

  Assuming R is not a process with a kell, nor a parallel composition of kell-m processes, we have already shown $\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[R] \,]\!] \,]\!] \,]\!] = \mathcal{V}[\![\, R \,]\!]$ in these cases, then:

$$\begin{array}{l}\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[\textbf{new a } R] \,]\!] \,]\!] \,]\!] \\ \quad = (\{\overline{K}\} \cup \mathcal{V}_M[\![\, \mathcal{I}[\![\, B \cup \{a\}, \mathcal{A}[\![\, K[R] \,]\!] \,]\!] \,]\!]) \setminus \{a\} \\ \quad = (\{\overline{K}\} \cup \mathcal{V}[\![\, R \,]\!]) \setminus \{a\} \\ \quad = \{\overline{K}\} \cup \mathcal{V}[\![\, \textbf{new}\ a\ R \,]\!]\end{array}$$

When Q is the parallel composition of processes R and T, and these processes are not kells, it can also be shown the kell passivation code S generates the null process **0**, and the only interesting remaining process is the advance-kell execution process. In such a case, showing that $\mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[R|T]\ ]\!]\ ]\!]\ ]\!] \cup \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|R]\ ]\!]\ ]\!]\ ]\!]$ = $\mathcal{V}[\![\ Q\ ]\!]$, requires computing the visibility predicates for the composed processes in Q. The procedure to follow is similar to the one just followed, and hence we skip it.

When Q has the form **new** a_1 **new** a_2 ...**new** a_n R, the restricted names are lifted from within the kell, and the resulting visibility expression is:

$$\begin{aligned}
&\mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[\textbf{new}\ a_1, ..., a_n\ \ R]\ ]\!]\ ]\!]\ ]\!] \\
&\quad = \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \textbf{new}\ a1, ..., a_n\ \mathcal{A}[\![\ K[R]\ ]\!]\ ]\!]\ ]\!] \\
&\quad = \mathcal{V}_M[\![\ \textbf{new}\ a_1, ..., a_n\ \mathcal{I}[\![\ B \cup \textbf{new}\ a_1, ..., a_n\ , \mathcal{A}[\![\ K[R]\ ]\!]\ ]\!]\ ]\!] \\
&\quad = (\{\overline{K}\} \cup \mathcal{V}[\![\ R\ ]\!]) \setminus \{a_1, ..., a_n\} \\
&\quad = \{\overline{K}\} \cup \mathcal{V}[\![\ \textbf{new}\ a_1, ..., a_n\ R\ ]\!]
\end{aligned}$$

We still need to show that when Q is itself a kell, then

$$\{\overline{K}\} \cup \mathcal{V}_M[\![\ \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!]\ ]\!] \cup \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]\ ]\!] = \{\overline{K}\} \cup \mathcal{V}[\![\ Q\ ]\!]$$

We argue that in the case of nested kells, the suspend kell $\mathcal{S}$ definition generates null processes for each non-kell subprocess, and a concretion for each subkell, plus the result of advancing the execution of the kell. We will see that the resulting visibility predicates match. Let us assume $P = K[K_1[...[K_n[R]]...]]$, with R a non-kell process (i.e., R has a form different than $L[\ .]$). In such a process, we have $Q = K_1[...[K_n[R]]...]$, and:

$$\begin{aligned}
&\{\overline{K}\} \cup \mathcal{V}_M[\![\ \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!]\ ]\!] \cup \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]\ ]\!] = \\
&\{\overline{K}\} \cup \mathcal{V}_M[\![\ \mathit{Hwrite}(\overline{K_1}(K_2[K_3[...[K_n[R]]...]], \{K\}).\mathcal{I}[\![\ B, K[\mathbf{0}]\ ]\!], B \setminus \{K, K_1\})\ ]\!] \\
&\quad \cup \mathcal{V}_M[\![\ \mathit{Hwrite}(\overline{K_2}(K_3[...[K_n[R]]...], \{K, K_1\}). \\
&\qquad\qquad \mathcal{I}[\![\ B, K[K_1[\mathbf{0}]]\ ]\!], B \setminus \{K, K_1, K_2\})\ ]\!] \\
&\quad \cup \cdots \\
&\quad \cup \mathcal{V}_M[\![\ \mathit{Hwrite}(\overline{K_n}(R, \{K, K_1, K_2, ..., K_{n-1}\}). \\
&\qquad\qquad \mathcal{I}[\![\ B, K[K_1[K_2[...[K_{n-1}[\mathbf{0}]]...]]]\ ]\!], B \setminus \{K, K_1, ..., K_n\})\ ]\!] \\
&\quad \cup \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[K_1[...[K_n[R]]...]]\ ]\!]\ ]\!]\ ]\!] = \\
&\{\overline{K}\} \cup \{\overline{K_1}\} \cup \{\overline{K_2}\} \cup \cdots \cup \{\overline{K_n}\} \cup \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, K[K_1[...[\mathcal{A}[\![\ K_n[R]\ ]\!]...]]\ ]\!]\ ]\!]
\end{aligned}$$

The expression $\mathcal{I}[\![\ B, K[K_1[...[\mathcal{A}[\![\ K_n[R]\ ]\!]...]]\ ]\!]$, by $\mathcal{I}$'s and $\mathcal{S}$'s definitions, takes the result of $\mathcal{A}[\![\ K_n[R]\ ]\!]$ and, on each recursive invocation of $\mathcal{A}$, moves the expression one kell outwards. For example, if $R = a(c) \triangleright R'$:

$\mathcal{A}$ inv.	Expression
0	$\mathcal{A}[\![\ K[\mathcal{A}[\![\ K_1[\mathcal{A}[\![\ K_2[...[\mathcal{A}[\![\ K_{n-1}[\mathcal{A}[\![\ K_n[a(c) \triangleright R']\ ]\!]]\ ]\!]...]]\ ]\!]]\ ]\!]]\ ]\!]\ ]\!]$
1	$\mathcal{A}[\![\ K[\mathcal{A}[\![\ K_1[\mathcal{A}[\![\ K_2[...[\mathcal{A}[\![\ K_{n-1}[a(c, \{K_n\}) \triangleright K_n[R']]\ ]\!]...]\ ]\!]]\ ]\!]]\ ]\!]\ ]\!]$
2	$\mathcal{A}[\![\ K[\mathcal{A}[\![\ K_1[\mathcal{A}[\![\ K_2[...[a(c, \{K_n, K_{n-1}\}) \triangleright K_{n-1}[K_n[R']]]...]\ ]\!]]\ ]\!]]\ ]\!]\ ]\!]$
...	...
$n+1$	$a(c, \{K_n, K_{n-1}, ..., K_1, K\}) \triangleright K[K_1[K_2[...[K_n[R']]...]]]$

Therefore:

$$\begin{aligned}
&\mathcal{V}_M[\![\ \mathcal{I}[\![\ B, K[K_1[...[\mathcal{A}[\![\ K_n[R]\ ]\!]...]]\ ]\!]\ ]\!] \\
&\qquad = \{K, K_1, K_2, ..., K_n\} \cup \mathcal{V}_M[\![\ \mathcal{A}[\![\ K_n[R]\ ]\!]\ ]\!]
\end{aligned}$$

Finally, by $\mathcal{V}$'s definition:

$$\mathcal{V}[\![\ K_1[K_2[...[K_n[R]]...]]\ ]\!] = \{K_1, K_2, ..., K_n\} \cup \mathcal{V}[\![\ R\ ]\!]$$

We have previously shown, for non-kell processes R, $\mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K_n[R]\ ]\!]\ ]\!]\ ]\!] = \mathcal{V}[\![\ R\ ]\!]$. Now, we have all the pieces:

$$\begin{aligned}
&\{\overline{K}\} \cup \mathcal{V}_M[\![\ \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!]\ ]\!] \cup \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]\ ]\!] \\
&\quad = \{\overline{K}\} \cup \{K, K_1, ..., K_n\} \cup \mathcal{V}_M[\![\ \mathcal{A}[\![\ K_n[R]\ ]\!]\ ]\!] \\
&\quad = \{K, K_1, ..., K_n\} \cup \mathcal{V}_M[\![\ \mathcal{A}[\![\ K_n[R]\ ]\!]\ ]\!] \\
&\quad = \{K\} \cup \mathcal{V}[\![\ K_1[K_2[...[K_n[R]]...]]\ ]\!]
\end{aligned}$$

With this, the last case for $K[Q]$, we have shown that $\mathcal{V}[\![\ K[Q]\ ]\!] = \mathcal{V}_M[\![\ \mathcal{I}[\![\ B, K[Q]\ ]\!]\ ]\!]$.

- $Q|R$, then $\mathcal{V}[\![\, P|Q \,]\!] \stackrel{def}{=} \mathcal{V}[\![\, P \,]\!] \cup \mathcal{V}[\![\, Q \,]\!]$. By $\mathcal{I}$'s and $\mathcal{V}_M$'s definitions, we have $\mathcal{I}[\![\, B, Q|R \,]\!] \stackrel{def}{=} \mathcal{I}[\![\, B, Q \,]\!] \mid \mathcal{I}[\![\, B, R \,]\!]$, and $\mathcal{V}_M[\![\, \mathcal{I}[\![\, B, Q \,]\!] | \mathcal{I}[\![\, B, R \,]\!] \,]\!] \stackrel{def}{=} \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, Q \,]\!] \,]\!] \cup \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, R \,]\!] \,]\!]$. Finally, by structural induction, $\mathcal{V}[\![\, Q \,]\!] = \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, Q \,]\!] \,]\!]$, and $\mathcal{V}[\![\, R \,]\!] = \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, R \,]\!] \,]\!]$; therefore, $\mathcal{V}[\![\, P|Q \,]\!] = \mathcal{V}_M[\![\, \mathcal{I}[\![\, B, Q|R \,]\!] \,]\!]$.

We have shown that, initially, P and its encoding $\mathcal{I}[\![\, \{\}, P \,]\!]$ have the same visibility predicates. By using LTS semantics, we will now show that if $P \rightarrow P'$, then $\mathcal{I}[\![\, B, P \,]\!] \rightarrow \mathcal{I}[\![\, B'', P'' \,]\!]$, with P'' structurally equivalent to P': $P'' \equiv P'$. For the kell-m calculus we will use the LTS semantics specified in Section 3.1. For MMCπ we use the LTS semantics from Section 5.6. We will only show the L-* version of the transition rules.

In the discussion below, K_c and K_a may be used for kell containment sets and for variables. If an abstraction K_a is the kell containment set for the abstraction and K_c is a variable, to be instantiated with the kell containment set of a matching concretion in τ transitions. In concretions K_c is the kell containment set of the concretion and K_a is the variable.

- OUT. The encoding for $\overline{a}(\widetilde{w})$ is:

$$\begin{aligned}\mathcal{I}[\![\, B, \overline{a}(\widetilde{w}) \,]\!] &\stackrel{def}{=} \mathit{Hwrite}(\overline{a}(\widetilde{w}, \emptyset), \mathbf{0}, B \setminus \{a\}) \\ &\equiv \textbf{new}\ \widetilde{h}\ \overline{a}(\widetilde{w}', K_a, \emptyset, B \setminus \{a\}).\mathbf{0}\end{aligned}$$

  With $\widetilde{h}$ the higher-order indicators for the higher-order expressions in $\widetilde{w}$. $\widetilde{w}'$ is $\widetilde{w}$ with higher-order expressions replaced by their higher-order indicators $h_i \in \widetilde{h}$. The kell-m transition is:

$$\overline{a}(\widetilde{w}) \xrightarrow{\overline{a}(\widetilde{w})} \mathbf{0}$$

  πPRE is the corresponding MMCπ transition:

$$\textbf{new}\ \widetilde{h}\ \overline{a}(\widetilde{w}', K_a, \emptyset, B \setminus \{a\}) \xrightarrow{0, \overline{a}(\widetilde{w}'', K_a, \emptyset, B')} \mathbf{0}$$

  $\widetilde{w}''$ is $\widetilde{w}'$ with higher-order indicators h_j replaced by **new** h_j; and B' is $B \setminus \{a\}$ with all $b_i \in B \setminus \{a\}$ replaced by **new** b_i. Therefore, trivially:

$$\mathcal{I}[\![\, B, \overline{a}(\widetilde{w}) \,]\!] \xrightarrow{0, \overline{a}(\widetilde{w}'', K_a, \emptyset, B')} \mathcal{I}[\![\, B, \mathbf{0} \,]\!]$$

- IN. The encoding for $a(\widetilde{c}) \triangleright Q$ is:

$$\begin{aligned}\mathcal{I}[\![\, B, a(\widetilde{c}) \triangleright Q \,]\!] &\stackrel{def}{=} a(\widetilde{c}, \emptyset, K_c, \mathit{bnd}).\mathtt{code}(\mathtt{inst}(Q, \widetilde{c}, \mathit{bnd}, Q_\pi), Q_\pi) \\ &\equiv a(\widetilde{c}, \mathit{bnd}).\mathcal{I}[\![\, B \cup \mathit{bnd}, \mathcal{F}[\![\, \mathcal{H}[\![\, Q \,]\!] \,]\!] \,]\!]\end{aligned}$$

  The result of the kell-m transitions is:

$$a(\widetilde{x}) \triangleright Q \xrightarrow{a(\widetilde{c})} Q$$

  The corresponding MMCπ transition is, once again, πPRE:

$$\begin{aligned}&a(\widetilde{c}, \emptyset, K_c, \mathit{bnd}).\mathcal{I}[\![\, B \cup \mathit{bnd}, \mathcal{F}[\![\, \mathcal{H}[\![\, Q \,]\!] \,]\!] \,]\!] \xrightarrow{0, a(\widetilde{c}, K_c, \emptyset, \mathit{bnd})} \\ &\quad \mathcal{I}[\![\, B \cup \mathit{bnd}, \mathcal{F}[\![\, \mathcal{H}[\![\, Q \,]\!] \,]\!] \,]\!]\end{aligned}$$

  Since there is no communication yet, $\mathcal{H}[\![\, Q \,]\!] = Q$. $\mathcal{F}$ is used to avoid unintended capturing of names in process expressions. The result of $\mathcal{F}$ on a process is structurally equivalent to the original process, therefore, $\mathcal{F}[\![\, \mathcal{H}[\![\, Q \,]\!] \,]\!] \equiv Q$, and:

$$\mathcal{I}[\![\, B, a(\widetilde{c}) \triangleright Q \,]\!] \xrightarrow{0, a(\widetilde{c}, \emptyset, K_c, \mathit{bnd})} \mathcal{I}[\![\, B \cup \mathit{bnd}, Q \,]\!]$$

- KELLOUT. The encoding for $K[Q]$, assuming Q is not the parallel composition of processes T and R, is:

$$\begin{aligned}\mathcal{I}[\![\, B, K[Q] \,]\!] &\stackrel{def}{=} \mathcal{S}[\![\, \mathbf{0}, B, K[Q], \emptyset \,]\!] + \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q] \,]\!] \,]\!] \\ &\equiv (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \\ &\quad \mathcal{S}[\![\, K[Q], B, Q, \{K\} \,]\!] + \mathcal{I}[\![\, B, \mathcal{A}[\![\, K[Q] \,]\!] \,]\!]\end{aligned}$$

Where h_Q is the higher-order indicator for Q. The transition for kell-m is:

$$K[Q] \xrightarrow{\overline{K}[Q]} \mathbf{0}$$

Notice, in MMCπ, $\overline{K}(h)$ is a concretion, and by πSUM:

$$\begin{array}{l}(\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0})\ +\ \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] + \\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!] \xrightarrow{0,\overline{K}(\textbf{new}\ h_Q\ ,K_a,\emptyset,B')} \mathbf{0}\end{array}$$

B' is $B \setminus \{K\}$, with $b_i \in B \setminus \{K\}$ replaced by **new** b_i . Since $\mathcal{I}[\![\ B, \mathbf{0}\ ]\!] \equiv \mathbf{0}$, then:

$$\mathcal{I}[\![\ B, K[Q]\ ]\!] \xrightarrow{0,\overline{K}(\textbf{new}\ h_Q\ ,K_a,\emptyset,B')} \mathcal{I}[\![\ B, \mathbf{0}\ ]\!]$$

When $Q \equiv R|T$, we have:

$$\begin{array}{rcl}\mathcal{I}[\![\ B, K[Q]\ ]\!] & \stackrel{def}{=} & \mathcal{S}[\![\ \mathbf{0}, B, K[Q], \emptyset\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[R|T]\ ]\!]\ ]\!] + \\ & & \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|R]\ ]\!]\ ]\!] \\ & \equiv & (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] + \\ & & \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[R|T]\ ]\!]\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|R]\ ]\!]\ ]\!]\end{array}$$

Again, by πSUM,

$$\mathcal{I}[\![\ B, K[Q]\ ]\!] \xrightarrow{0,\overline{K}(\textbf{new}\ h_Q\ \ K_a,\emptyset,B')} \mathcal{I}[\![\ B, \mathbf{0}\ ]\!]$$

- KELLIN. The encoding for $K[x] \triangleright Q$ is:

$$\begin{array}{rcl}\mathcal{I}[\![\ B, K[x] \triangleright Q\ ]\!] & \stackrel{def}{=} & \mathcal{I}[\![\ B, K(x, \emptyset) \triangleright Q\ ]\!] \\ & \equiv & K(x, \emptyset, K_c, bnd).\mathcal{I}[\![\ B \cup bnd, \mathcal{F}[\![\ \mathcal{H}[\![\ Q\ ]\!]\ ]\!]\ ]\!]\end{array}$$

The transition for kell-m is:

$$K[x] \xrightarrow{K[x]} Q$$

Using the πPRE transition for MMCπ we obtain:

$$\begin{array}{l}K(x, \emptyset, K_c, bnd).\mathcal{I}[\![\ B \cup bnd, \mathcal{F}[\![\ \mathcal{H}[\![\ Q\ ]\!]\ ]\!]\ ]\!] \xrightarrow{0,K(x,\emptyset,K_c,bnd)} \\ \quad \mathcal{I}[\![\ B \cup bnd, \mathcal{F}[\![\ \mathcal{H}[\![\ Q\ ]\!]\ ]\!]\ ]\!]\end{array}$$

Since no communication has taken place, $Q \equiv \mathcal{F}[\![\ \mathcal{H}[\![\ Q\ ]\!]\ ]\!]$, therefore,

$$\mathcal{I}[\![\ B, K[x] \triangleright Q\ ]\!] \xrightarrow{0,K(x,\emptyset,K_c,bnd)} \mathcal{I}[\![\ B \cup bnd, Q\ ]\!]$$

- RESTRICT. We have $Q \xrightarrow{\alpha} R$, and we will assume, by structural induction, $\mathcal{I}[\![\ B, Q\ ]\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![\ B, R\ ]\!]$. We need to show that $\mathcal{I}[\![\ B, \textbf{new}\ c\ Q\ ]\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![\ B, \textbf{new}\ c\ R\ ]\!]$, with $c \notin bn(\alpha')$. By $\mathcal{I}$'s definition:

$$\mathcal{I}[\![\ B, \textbf{new}\ c\ Q\ ]\!] \stackrel{def}{=} \textbf{new}\ c\ \mathcal{I}[\![\ B, Q\ ]\!]$$

By πRES, when $c \notin names(M, \alpha')$,

$$\textbf{new}\ c\ \mathcal{I}[\![\ B, Q\ ]\!] \xrightarrow{M,\alpha'} \textbf{new}\ c\ \mathcal{I}[\![\ B, R\ ]\!]$$

And, since **new** $c\ \mathcal{I}[\![\ B, R\ ]\!] \equiv \mathcal{I}[\![\ B, \textbf{new}\ c\ R\ ]\!]$, we have

$$\mathcal{I}[\![\ B, \textbf{new}\ c\ Q\ ]\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![\ B, \textbf{new}\ c\ R\ ]\!]$$

- PAR. We have $Q \xrightarrow{\alpha} R$ and, by structural induction, assume $\mathcal{I}[\![B_Q, Q]\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![B_R, R]\!]$. We need to show that $\mathcal{I}[\![B_Q, Q|Q']\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![B_R, R|Q']\!]$ when $\alpha' \notin \mathit{fn}(Q')$. By πPAR:

$$\mathcal{I}[\![B_Q, Q]\!] \mid \mathcal{I}[\![B_Q, Q']\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![B_R, R]\!] \mid \mathcal{I}[\![B_R, Q']\!]$$

  By $\mathcal{I}'s$ definition, $\mathcal{I}[\![B_Q, Q]\!] \mid \mathcal{I}[\![B_Q, Q']\!] \equiv \mathcal{I}[\![B_Q, Q|Q']\!]$. Therefore,

$$\mathcal{I}[\![B_Q, Q|Q']\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![B_R, R|Q']\!]$$

- OPEN. We have $Q \xrightarrow{\overline{a}(\widetilde{w})} R$, and we have already shown:

$$\mathcal{I}[\![B_Q, Q]\!] \xrightarrow{0,\overline{a}(\widetilde{w}'', K_a, K_c, B')} \mathcal{I}[\![B_R, R]\!]$$

  By πOPEN:

$$\textbf{new } c\, \mathcal{I}[\![B_Q, Q]\!] \xrightarrow{0,\overline{a}(\widetilde{u}, K_a, K_c, B')} \mathcal{I}[\![B_R, R]\!]$$

  With $\widetilde{u}$ as $\widetilde{w}''$, but with c replaced by **new** c . By $\mathcal{I}$'s definition, we have **new** $c\, \mathcal{I}[\![B_Q, Q]\!] \equiv \mathcal{I}[\![B_Q, \textbf{new } c\ Q]\!]$, therefore,

$$\mathcal{I}[\![B_Q, \textbf{new } c\ Q]\!] \xrightarrow{0,\overline{a}(\widetilde{u}, K_a, K_c, B')} \mathcal{I}[\![B_R, R]\!]$$

- L-REACT and L-CLOSE. We have $Q \xrightarrow{a(\widetilde{c})} Q'$ and $R \xrightarrow{\overline{a}(\widetilde{w})} R'$. By structural induction we assume,

$$\mathcal{I}[\![B_Q, Q]\!] \xrightarrow{M,a(\widetilde{c}, K_a, K_c, bnd)} \mathcal{I}[\![B'_Q, Q']\!]$$

  and,

$$\mathcal{I}[\![B_R, R]\!] \xrightarrow{N,\overline{a}(\widetilde{w}'', K_a, K_c, B')} \mathcal{I}[\![B'_R, R']\!]$$

  By πCOM and πCLOSE, depending on whether or not there are restricted names or higher-order indicators being passed in the communication, we have:

$$\begin{aligned}&\mathcal{I}[\![B_Q, Q]\!] \mid \mathcal{I}[\![B_R, R]\!] \\ &\quad \xrightarrow{M\cup N,\tau} \textbf{new } \widetilde{d}\ (\mathcal{I}[\![B'_Q, Q']\!]\{(\widetilde{w}'', B')/(\widetilde{c}, bnd)\} \mid \mathcal{I}[\![B'_R, R']\!])\end{aligned}$$

  where $\widetilde{d}$ are the restricted names being passed in the communication (if any). Finally, by $\mathcal{I}$'s definition, we have:

$$\begin{aligned}&\mathcal{I}[\![B_Q \cup B_R, Q|R]\!] \\ &\quad \xrightarrow{M\cup N,\tau} \mathcal{I}[\![B'_Q \cup B'_R, \textbf{new } \widetilde{d}\ (Q'\{(\widetilde{w}'', B')/(\widetilde{c}, bnd)\} \mid R')]\!]\end{aligned}$$

  Note by $\mathcal{I}$'s definition, $\mathcal{I}[\![B'_Q, Q']\!]$ has the form $\mathcal{I}[\![B'_Q, \mathcal{F}[\![\mathcal{H}[\![\ldots]\!]]\!]]\!]$. This guarantees that, on communication, higher-order indicators are replaced by their associated higher-order expressions.

- L-SUSPEND. We have $Q \xrightarrow{K[x]} Q'$ and $R \xrightarrow{\overline{K}(T)} R'$, where T is the process inside kell K. By structural induction we assume,

$$\mathcal{I}[\![B_Q, Q]\!] \xrightarrow{M,K(x,K_a,K_c,bnd)} \mathcal{I}[\![B'_Q, Q']\!]$$

  And,

$$\mathcal{I}[\![B_R, R]\!] \xrightarrow{N,\overline{K}(\textbf{new } h_T\ ,K_a,K_c,B')} \mathcal{I}[\![B'_R, R']\!]$$

  Where h_T is the higher-order indicator for T. Since we use channels to represent kells, using πCLOSE we obtain:

$$\begin{aligned}&\mathcal{I}[\![B_Q, Q]\!] \mid \mathcal{I}[\![B_R, R]\!] \\ &\quad \xrightarrow{M\cup N,\tau} \textbf{new } h_T, \widetilde{d}\ (\mathcal{I}[\![B'_Q, Q']\!]\{(h_T, B')/(x, bnd)\} \mid \mathcal{I}[\![B'_R, R']\!])\end{aligned}$$

Where $\widetilde{d}$ are the restricted names, with the exception of h_T, being passed in the communication (if any). Using $\mathcal{I}$'s definition, we obtain:

$$\begin{array}{l}\mathcal{I}[\![\ B_Q \cup B_R, Q|R\ ]\!] \\ \qquad \xrightarrow{M\cup N,\tau} \mathcal{I}[\![\ B'_Q \cup B'_R, \textbf{new}\ h_T, \widetilde{c}\ (Q'\{(h_T, B')/(x, bnd)\} \mid R')\ ]\!]\end{array}$$

As with rules L-REACT and L-CLOSE, by $\mathcal{I}$'s definition, $\mathcal{I}[\![\ B'_Q, Q'\ ]\!]$ has the form $\mathcal{I}[\![\ B'_Q, \mathcal{F}[\![\ \mathcal{H}[\![\ \ldots\ ]\!]\ ]\!]\ ]\!]$, guaranteeing that the higher-order indicator h_T is replaced by its associated higher order expression T.

- ADVANCE. When $Q \xrightarrow{\alpha} R$, then $K[Q] \xrightarrow{\alpha} K[R]$. By structural induction, we assume

$$\mathcal{I}[\![\ B, Q\ ]\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![\ B', R\ ]\!]$$

As we have seen,

$$\begin{array}{rl}\mathcal{I}[\![\ B, K[Q]\ ]\!] \equiv & \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0} + \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] + \\ & \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!]\end{array}$$

Or,

$$\begin{array}{rl}\mathcal{I}[\![\ B, K[Q]\ ]\!] \equiv & \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0} + \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] + \\ & \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[U|T]\ ]\!]\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[T|U]\ ]\!]\ ]\!]\end{array}$$

If $Q \equiv U|T$. In general, Q can have one of the following forms:

- $a(\widetilde{c}) \triangleright R$:

$$\begin{array}{l}(\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] + \\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[Q]\ ]\!]\ ]\!] \\ \quad \equiv\ (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] + \\ \qquad \mathcal{I}[\![\ B, a(\widetilde{c}) \triangleright K[R]\ ]\!] \\ \quad \equiv\ (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[Q], B, Q, \{K\}\ ]\!] + \\ \qquad a(\widetilde{c}, \{K\}, K_c, bnd).\mathcal{I}[\![\ B \cup bnd, \mathcal{F}[\![\ \mathcal{H}[\![\ \mathrm{K[R]}\ ]\!]\ ]\!]\ ]\!] \xrightarrow{0,a(\widetilde{c},\{K\},K_c,bnd)} \\ \qquad \mathcal{I}[\![\ B \cup bnd, \mathcal{F}[\![\ \mathcal{H}[\![\ \mathrm{K[R]}\ ]\!]\ ]\!]\ ]\!]\end{array}$$

  Since no communication has yet taken place, $\mathcal{F}[\![\ \mathcal{H}[\![\ \mathrm{K[R]}\ ]\!]\ ]\!] \equiv K[R]$. Therefore,

$$\mathcal{I}[\![\ B, K[Q]\ ]\!] \xrightarrow{0,a(\widetilde{c},\{K\},K_c,bnd)} \mathcal{I}[\![\ B \cup bnd, K[R]\ ]\!]$$

- $\overline{a}(\widetilde{w})$. In this case $R = \mathbf{0}$, and

$$\begin{array}{l}(\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[\overline{a}(\widetilde{w})], B, \overline{a}(\widetilde{w}), \{K\}\ ]\!] + \\ \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[\overline{a}(\widetilde{w})]\ ]\!]\ ]\!] \\ \quad \equiv\ (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[\overline{a}(\widetilde{w})], B, \overline{a}(\widetilde{w}), \{K\}\ ]\!] + \\ \qquad \mathcal{I}[\![\ B, \overline{a}(\widetilde{w})\ ]\!] \\ \quad \equiv\ (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[\overline{a}(\widetilde{w})], B, \overline{a}(\widetilde{w}), \{K\}\ ]\!] + \\ \qquad \textbf{new}\ \widetilde{h}\ \overline{a}(\widetilde{w}', \emptyset, B \setminus \{a\}).\mathbf{0} \xrightarrow{0,\overline{a}(\widetilde{w}'',K_a,\{K\},B'')} \mathbf{0}\end{array}$$

  With $\widetilde{h}$ the higher-order indicators for the higher-order expressions in $\widetilde{w}$. $\widetilde{w}'$ is $\widetilde{w}$ with higher-order expressions replaced by their higher-order indicators $h_i \in \widetilde{h}$. $\widetilde{w}''$ is $\widetilde{w}'$, with the higher-order indicators, h_i, replaced with **new** h_i ; finally, B'' is $B \setminus \{a\}$ with all $b_i \in B \setminus \{a\}$ replaced by **new** b_i. Since $\mathcal{I}[\![\ B, \mathbf{0}\ ]\!] \equiv \mathbf{0}$:

$$\mathcal{I}[\![\ B, K[\overline{a}(\widetilde{w})]\ ]\!] \xrightarrow{0,\overline{a}(\widetilde{w}'',K_a,\{K\},B'')} \mathcal{I}[\![\ B \cup bnd, K[0]\ ]\!]$$

– $L[x] \triangleright T$. In this case $R = T$, and

$$
\begin{array}{l}
(\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[L[x] \triangleright T], B, L[x] \triangleright T\ ]\!] + \\
\mathcal{I}[\![\ B, \mathcal{A}[\![\ K[L[x] \triangleright T]\ ]\!]\ ]\!] \\
\quad \equiv\ (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \\
\quad\quad \mathcal{S}[\![\ K[L[x] \triangleright T], B, L[x] \triangleright T, \{K\}\ ]\!] + \mathcal{I}[\![\ B, L(x) \triangleright K[T]\ ]\!] \\
\quad \equiv\ (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \\
\quad\quad \mathcal{S}[\![\ K[L[x] \triangleright T], B, L[x] \triangleright T, \{K\}\ ]\!] + \\
\quad\quad L(x, \{K\}, K_c, bnd).\mathcal{I}[\![\ B \cup bnd, \mathcal{F}[\![\ \mathcal{H}[\![\ K[T]\ ]\!]\ ]\!]\ ]\!] \xrightarrow{0, L(x, \{K\}, K_c, bnd)} \\
\quad\quad \mathcal{I}[\![\ B \cup bnd, \mathcal{F}[\![\ \mathcal{H}[\![\ K[T]\ ]\!]\ ]\!]\ ]\!]
\end{array}
$$

Since no communication has yet taken place, $\mathcal{F}[\![\ \mathcal{H}[\![\ K[T]\ ]\!]\ ]\!] \equiv K[T]$, and:

$$\mathcal{I}[\![\ B, K[L[x] \triangleright T]\ ]\!] \xrightarrow{0, L(x, \{K\}, K_c, bnd)} \mathcal{I}[\![\ B \cup bnd, K[T]\ ]\!]$$

– $L[T]$. In this case $R = \mathbf{0}$, or $R = L[U]$. If $R = \mathbf{0}$ we have:

$$
\begin{array}{l}
(\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \mathcal{S}[\![\ K[L[T]], B, L[T], \{K\}\ ]\!] + \\
\mathcal{I}[\![\ B, \mathcal{A}[\![\ K[L[T]]\ ]\!]\ ]\!] \\
\quad \equiv\ (\textbf{new}\ h_Q\ \overline{K}(h_Q, K_a, \emptyset, B \setminus \{K\}).\mathbf{0}) + \\
\quad\quad (\textbf{new}\ h_T\ \overline{L}(h_T, K_a, \{K\}, B \setminus \{L\}).\mathcal{I}[\![\ N, K[\mathbf{0}]\ ]\!]) + \\
\quad\quad \mathcal{S}[\![\ K[L[T]], B, T, \{K, L\}\ ]\!] + \mathcal{I}[\![\ B, \mathcal{A}[\![\ K[L[T]]\ ]\!]\ ]\!] \\
\quad\quad \xrightarrow{0, \overline{L}(h_T, K_a, \{K\}, B'')} \mathcal{I}[\![\ N, K[\mathbf{0}]\ ]\!]
\end{array}
$$

Where h_T is the higher-order indicator for T, and B'' is $B \setminus \{L\}$ with all $b_i \in B \setminus \{L\}$ replaced by **new** b_i. Since $T = \mathbf{0}$

$$\mathcal{I}[\![\ B, K[L[T]]\ ]\!] \xrightarrow{0, \overline{L}(h_T, K_a, \{K\}, B'')} \mathcal{I}[\![\ B, K[T]\ ]\!]$$

If $R = L[U]$ we have instead, $L[T] \xrightarrow{\alpha} L[U]$. This is the case where a subkell in T is suspended, or $L[T]$ advances its execution. Let us assume that a subkell K_i in T is suspended, and T being composed of n subkells $K_1, K_2, ..., K_n$. The encoding of $L[T]$ is a process with the following structure:

$$
\begin{array}{l}
\textbf{new}\ h_1\ \overline{K_1}(h_1, \{K\}, B \setminus \{K_1\}).\mathcal{I}[\![\ ...\ ]\!] + \\
\textbf{new}\ h_2\ \overline{K_2}(h_2, \{K, K_1\}, B \setminus \{K_2\}).\mathcal{I}[\![\ ...\ ]\!] + \\
... + \\
\textbf{new}\ h_n\ \overline{K_n}(h_n, \{K, K_1, K_2, ..., K_{n-1}\}, B \setminus \{K_n\}).\mathcal{I}[\![\ ...\ ]\!] + ...
\end{array}
$$

Where h_i is the higher-order indicator for the process inside the i-th subkell. By πSUM, we have:

$$\mathcal{I}[\![\ B, K[L[T]]\ ]\!] \xrightarrow{0, \overline{K_i}(h_i, K_a, \{K, K_1, K_2, ..., K_{i-1}\}, B')} \mathcal{I}[\![\ B, K[U]\ ]\!]$$

If $L[T]$ is advancing its execution we need to show:

$$\mathcal{I}[\![\ B, \mathcal{A}[\![\ K[L[T]]\ ]\!]\ ]\!] \xrightarrow{M, \alpha'} \mathcal{I}[\![\ B, K[L[U]]\ ]\!]$$

Note $\mathcal{A}$ lifts concretions and abstractions from the inner kells towards the outside kells, eventually obtaining expressions $a(\widetilde{c}) \triangleright K[...]$, $\overline{a}(\widetilde{w})\,|K[...]$, and $K'[x] \triangleright K[...]$. Once the concretion or abstraction corresponding to α is outside K, the resulting process is encoded into a MMCπ expression on which, as we have shown, a corresponding MMCπ transition can be applied.

– $U|T$, with U or T having one of the forms already considered. Let us assume U is the process transitioning from U to U'. We have already shown $\mathcal{I}[\![\ B, U\ ]\!] \xrightarrow{M, \alpha'} \mathcal{I}[\![\ B, U'\ ]\!]$. Since T stays the same, we have:

$$\mathcal{I}[\![\ B, U|T\ ]\!] \stackrel{def}{=} \mathcal{I}[\![\ B, U\ ]\!]|\mathcal{I}[\![\ B, T\ ]\!] \xrightarrow{M, \alpha'} \mathcal{I}[\![\ B, U'\ ]\!]|\mathcal{I}[\![\ B, T\ ]\!]$$

Since $\mathcal{I}[\![\ B, U'\ ]\!]|\mathcal{I}[\![\ B, T\ ]\!] \equiv \mathcal{I}[\![\ B, U'|T\ ]\!]$, then

$$\mathcal{I}[\![\ B, U|T\ ]\!] \xrightarrow{M,\alpha'} \mathcal{I}[\![\ B, U'|T\ ]\!]$$

The other case is when there is a communication between U and T. In this case $\alpha' = \tau$, and R is $U'|T'$. We have already shown for L-REACT, L-SUSPEND, and L-CLOSE:

$$\mathcal{I}[\![\ B, U|T\ ]\!] \xrightarrow{M,\tau} \mathcal{I}[\![\ B', U'|T'\ ]\!]$$

When $K[U|T]$, $\mathcal{A}$ non-deterministically advances the execution of U or T. Eventually, the interacting actions are lifted outside K, where they communicate.

6 Encoding kμ

the syntax of kμ is inspired by the implementation of the $\pi\mu$-calculus for the Mobility Model Checker (MMC) [17]. Systems are modelled in MMC using a variation of the first-order π-calculus. Since there is no locality in the π-calculus, the main difference between MMC and kμ, is the lack of kell and kell containment modalities in MMC. Also, because the first-order nature of its modelling formalism, properties in MMC can only be verified for first-order processes.

Formulas in MMC have the syntax specified in Figure 24. Possible actions α in MMCπ have the form τ, $\overline{a}(\widetilde{c})$, or $a(\widetilde{c})$. S is a set of π-calculus actions, $\mathcal{V}$ is a set of names, $\mathcal{N}$ is a set of formula names, and $\mathcal{Z}$ is a set of formula variables. Formulas can be named using the syntax $\texttt{fDef}(\mathcal{N}(\widetilde{\mathcal{V}}), \mathcal{F})$. Recursion is allowed in the definition of named formulas.

Diamond and box modalities are specified with $\texttt{fDiam}(S, \mathcal{F})$ and $\texttt{fBox}(S, \mathcal{F})$. Set, set minus, and not-action box and diamond modalities have similar syntax. Only named formulas can be negated $\texttt{neg\_form}(\mathcal{N}(\widetilde{\mathcal{V}}))$. `pred` is used to compare names.

$$\begin{array}{rl}\mathcal{F} ::= & \texttt{tt} \mid \texttt{ff} \mid \texttt{neg\_form}(\mathcal{N}(\widetilde{\mathcal{V}})) \mid \texttt{pred(Cond}, \mathcal{F}) \mid \texttt{fAnd}(\mathcal{F}, \mathcal{F}) \mid \texttt{fOr}(\mathcal{F}, \mathcal{F}) \mid \\ & \texttt{fDiam}(\alpha, \mathcal{F}) \mid \texttt{fDiamMinus}(\alpha, \mathcal{F}) \mid \texttt{fDiamSet}(S, \mathcal{F}) \mid \\ & \texttt{fDiamSetMinus}(S, \mathcal{F}) \mid \texttt{fBox}(\alpha, \mathcal{F}) \mid \texttt{fBoxMinus}(\alpha, \mathcal{F}) \mid \texttt{fBoxSet}(S, \mathcal{F}) \mid \\ & \texttt{fBoxSetMinus}(S, \mathcal{F}) \mid \texttt{fDef}(\mathcal{N}(\widetilde{\mathcal{Z}}), \mathcal{F}) \mid \texttt{form}(\mathcal{N}(\widetilde{\mathcal{V}}))\end{array}$$

Figure 24: Syntax of Property Formulas in the Mobility Model Checker

As it is the case with kμ, in MMCπ process evolution paths are implicitly quantified. The quantification if *for all paths labelled with action* in the case of box modalities, and *in at least one path labelled with action* for diamond modalities.

For notational convenience we write $\mathcal{F} \wedge \mathcal{F}$ and $\mathcal{F} \vee \mathcal{F}$ instead of $\texttt{fAnd(}\mathcal{F}, \mathcal{F}\texttt{)}$ and $\texttt{fOr(}\mathcal{F}, \mathcal{F}\texttt{)}$. Similarly we write $\neg\mathcal{N}(\mathcal{V})$ for $\texttt{neg\_form(}\mathcal{N}(\mathcal{V})\texttt{)}$; $\mathcal{N}(\widetilde{\mathcal{Z}}) \stackrel{def}{=} \mathcal{F}$ for $\texttt{fDef}(\mathcal{N}(\widetilde{\mathcal{Z}}), \mathcal{F})$; $\langle\alpha\rangle.\mathcal{F}$ for $\texttt{fDiam}(\alpha, \mathcal{F})$; and $[\alpha].\mathcal{F}$ for $\texttt{fBox}(\alpha, \mathcal{F})$. Idem for set diamond and set box modalities.

6.1 Kell Containment Conditions

A kell containment condition γ in kμ has one of the following forms, where I is a variable and $\mathcal{K}$ is a set of kells (cf. Section 4.1) :

$$\gamma \quad ::= \quad * \mid \mathcal{K} \mid \supseteq \mathcal{K} \mid \not\subseteq \mathcal{K} \mid I$$

We start by defining, in Figure 25, function $\mathcal{E}_p$ for the translation of kell containment conditions. To avoid confusion with the functions defined in Section 5, the functions defined for the encoding of kμ have a suffix $_p$.

The function has as arguments a kμ kell containment condition γ and a list of kells K_s. K_s is the actual set of kells where a communication action is executing. The return value is a MMC formula of the form `tt` or `pred(Cond, tt)`.

When a variable I is specified it is instantiated with the set of kells K_s. The other possible values are directly deduced from the kμ semantics (cf. Section 4.2 and Figure 16).

$$
\begin{array}{rcl}
\mathcal{E}_p(I,\kappa) & \stackrel{def}{=} & \texttt{pred}(I := \kappa, \texttt{tt}) \\
\mathcal{E}_p(*,\kappa) & \stackrel{def}{=} & \texttt{tt} \\
\mathcal{E}_p(\mathcal{K},\kappa) & \stackrel{def}{=} & \texttt{pred}(\mathcal{K} = \kappa, \texttt{tt}) \\
\mathcal{E}_p(\supseteq \mathcal{K},\kappa) & \stackrel{def}{=} & \texttt{pred}(\mathcal{K} \setminus \kappa = \emptyset, \texttt{tt}) \\
\mathcal{E}_p(\not\subseteq \mathcal{K},\kappa) & \stackrel{def}{=} & \texttt{pred}(\mathcal{K} \cap \kappa = \emptyset, \texttt{tt})
\end{array}
$$

Figure 25: Encoding of Kell Containment Conditions

$$
\begin{array}{rcl}
\mathcal{A}_p(\overline{a}(\widetilde{w})) & \stackrel{def}{=} & \overline{a}(\widetilde{w}, K_a, K_c, B) \\
\mathcal{A}_p(a(\widetilde{c})) & \stackrel{def}{=} & a(\widetilde{c}, K_a, K_c, B) \\
\mathcal{A}_p(\overline{k}[h]) & \stackrel{def}{=} & \overline{k}(h, K_a, K_c, B) \\
\mathcal{A}_p(k[x]) & \stackrel{def}{=} & k(x, K_a, K_c, B) \\
\mathcal{A}_p(\overleftrightarrow{a}(\widetilde{w})) & \stackrel{def}{=} & \tau(\overline{a}(\widetilde{w}, K_a, K_c, B)) \\
\mathcal{A}_p(\overleftrightarrow{k}[h]) & \stackrel{def}{=} & \tau(\overline{k}(h, K_a, K_c, B))
\end{array}
$$

Figure 26: Encoding of Actions

For readability we use set operators in the definition. The implementation of $\mathcal{E}_p$ in the kell-m checker has the set operations, shown as arguments to predicate `pred`, replaced with the corresponding Prolog set predicates.

6.2 Actions

In Figure 26 we define $\mathcal{A}_p$, a function for the translation of kell-m actions into MMCπ actions. The function receives as its only argument an action, α or α_τ, and returns the corresponding MMC action. Recall α represents abstraction and concretion transitions; α_τ represents τ transitions (e.g., $\overleftrightarrow{a}(\widetilde{w})$).

Because kells are encoded in MMC using regular channels (cf. Section 5), kell abstractions and concretions in kell-m correspond to channel concretions in MMC. τ transitions are encoded as MMC τ transitions extended to expose the channel or kell involved in the communication and the parameters of the communication. In all MMC actions, kell containment sets K_a and K_c are included in the parameters of the communication as well as the set of bound names B. Sets are implemented as lists in the encoding.

For abstractions, K_c and B correspond to Prolog variables, and K_a to the set of kells where the abstraction is executing. For concretions K_a is a Prolog variable, K_c is the set of kells where the concretion is located, and B contains the bound names of the process.

When τ transitions, all K_a, K_c and B are sets. K_a is the kell containment set of the matching abstraction; K_c is the kell containment set of the matching concretion; and B is the set of bound names of the concretion.

As we show later, depending on the type action, the sets K_a and K_c are used as parameter K_s in $\mathcal{E}$, the encoding of kell containment conditions.

6.3 Formulas

$\mathcal{T}_p$, defined in Figure 27, is the encoding of kμ formulas into MMC formulas.

Diamond and set modalities may receive an action condition or a set of action conditions. Action conditions, δ in kμ have the form (α, γ) or $(\alpha_\tau, \gamma, \gamma)$. α and α_τ identify the actions of interest; γ impose kell containment requirements on the location of the actions specified by α and α_τ.

In MMC it is only possible to identify actions of interest. Therefore the kμ encoding lifts the kell containment conditions from the transitions into the formulas being checked. Notice, in Figure 27, the γ_π, corresponding to the MMC encoding of kell containment conditions γ, occurs in the formulas following the diamond and box modalities.

Kell-m actions and kell containment conditions are translated into MMC by $\mathcal{AE}_p$. Kell containment sets are specified in the result of $\mathcal{A}_p$. The sets are used by $\mathcal{E}_p$ in the encoding of kell containment conditions. Name a is used

to represent channel actions and name k is used to represent kell actions. When encoded in MMC, both a and k are represented as channel actions. Since MMC is first-order, name h is used as parameter in kell concretions to indicate a higher-order indicator (cf. Figure 18).

To prove the encoding is correct, it is necessary to demonstrate, for any kμ formula $\mathcal{F}$: (a) the result $\mathcal{T}_p(\mathcal{F})$ of the encoding is a MMC property; and (b) if $P \models_{\mathcal{V}} \mathcal{F}$ for an interpretation of formula parameters $\mathcal{V}$, then $\mathcal{I}[\![\ \emptyset, P\ ]\!] \models \mathcal{T}_p(\mathcal{F})$. (a) and (b) can be proved by structural induction.

We informally argue about the correctness of the encoding of $\langle\alpha,\gamma\rangle.\mathcal{F}$. A similar argument can be made for the other types of kμ formulas. Notice the actions returned by $\mathcal{A}_p$ are MMCπ actions, and therefore valid action specifications in diamond and box modalities within MMC properties. $\mathcal{E}_p$ returns either `tt` or a `pred` specification, both valid MMC properties. The MMC encoding of $\langle\alpha,\gamma\rangle.\mathcal{F}$ is, by definition:

$$\mathcal{T}_p(\langle\alpha,\gamma\rangle.\mathcal{F}) \stackrel{def}{=} \langle\alpha_\pi\rangle.(\gamma_\pi \wedge \mathcal{T}_p(\mathcal{F})), \text{ with } \mathcal{AE}((\alpha,\gamma)) = (\alpha_\pi,\gamma_\pi)$$

Assuming, by structural induction, that $\mathcal{T}_p(\mathcal{F})$ is a valid MMC property formula, then $\langle\alpha_\pi\rangle.(\gamma_\pi \wedge \mathcal{T}_p(\mathcal{F})$ is a valid MMC property formula.

According to kμ semantics, the meaning of $\langle\alpha,\gamma\rangle.\mathcal{F}$ is:

$$P \models_{\mathcal{V}} \langle\alpha,\gamma\rangle.\mathcal{F} \text{ when } \exists Q : P \xrightarrow{\alpha',\kappa} Q \ \wedge\ \mathit{cmp}(\alpha,\alpha') \ \wedge\ \mathit{kc}(\gamma,\kappa) \ \wedge\ Q \models_{\mathcal{V}} \mathcal{F}''$$

cmp, defined in Figure 16, decides if an action specification in a kμ formula matches an action in the extended LTS. *kc* decides if a kell containment condition holds for a given kell containment set. $\mathcal{F}''$ is $\mathcal{F}$ after alpha converting (replacing) any parameters in α with actual values used in the communication. In the MMC encoding we require a transition $P_\pi \xrightarrow{\alpha_\pi} Q_\pi$ after which both, the kell containment condition γ_π and the encoding of $\mathcal{F}$, must hold. P_π is the MMC process corresponding to P and Q_π to Q.

Let us assume the formula holds in kell-m but not in MMC. This means there is no transition $P_\pi \xrightarrow{\alpha_\pi} Q_\pi$ after which $Q_\pi \not\models (\gamma_\pi \wedge \mathcal{F}_p)$. This may happen only if there is no $P_\pi \xrightarrow{\alpha_\pi} Q_\pi$, or if $(\gamma_\pi \wedge \mathcal{F}_p)$ does not hold at Q_π.

Because a kell-m process and its MMC encoding are behaviourally equivalent (Appendix 5.7), such a transition $P_\pi \xrightarrow{\alpha_\pi} Q_\pi$ must exist. Therefore the only possibility is $(\gamma_\pi \wedge \mathcal{F}_p)$ does not hold at Q_π. This could happen if γ_π does not hold at Q_π, or if $\mathcal{F}_p$ does not hold at Q_π. Let us assume γ_π does not hold at Q_π. This implies the kell containment set, built for the action in the encoding, does not match the kell containment set as specified in the extended LTS semantics for P.

For kell abstractions and channel abstractions and concretions, $\mathcal{A}$ builds the kell containment set. For kell concretions $\mathcal{S}$ builds the kell containment set. Every time an action is lifted from a kell, $\mathcal{A}$ adds the action to the associated kell containment set (cf. Figure 21). When the passivation code for a kell is generated (cf. Figure 19), the kell containment set for the kell concretion is updated with kell information as passivation code is generated from the external kells to the nested kells. These kells are included as parameters in the actions when the actions are encoded in MMC.

Assuming the kell containment sets are properly built in the encoding, and also assuming the kell containment sets are available after the actions, the kell containment condition may still not hold if it is not properly encoded by $\mathcal{E}_p$. Since $\mathcal{E}_p$ implements *kc* of the kμ kell containment semantics (cf. Figure 16), the kell containment condition must hold. Consequently, the only remaining case is $\mathcal{F}_p$ not holding at Q_π. But we argue, by structural induction, if $\mathcal{F}$ is not a diamond modality, $Q_\pi \models \mathcal{F}_p$ and $P \models_{\mathcal{V}} \langle\alpha,\gamma\rangle.\mathcal{F} \Rightarrow P_\pi \models \mathcal{T}_p(\langle\alpha,\gamma\rangle.\mathcal{F})$. If $\mathcal{F}$ is a diamond modality, we apply the argument above as many times as necessary until the unfolding of $\mathcal{F}$ leads to a non-diamond formula, at which point we can argue by structural induction.

Conversely, $\mathcal{T}_p(\langle\alpha,\gamma\rangle.\mathcal{F})$ may hold for a process P_π, but $P \not\models_{\mathcal{V}} \langle\alpha,\gamma\rangle.\mathcal{F}$, where P is the kell-m process corresponding to P_π. Because of the behavioural equivalence of kell-m processes and their MMC encoding, this can only happen if the kell containment condition holds for a transition in the MMC process but not in the kell-m process, or if $\mathcal{F}$ holds after the transition in the MMC process but not in the kell-m process. Assuming kell containment sets and conditions are properly translated, if $\mathcal{F}$ is not a diamond modality we can argue, by structural induction, this cannot occur and $\mathcal{T}_p(\langle\alpha,\gamma\rangle.\mathcal{F}) \Rightarrow \langle\alpha,\gamma\rangle.\mathcal{F}$. If $\mathcal{F}$ is a diamond modality we need to, once again, unfold $\mathcal{F}$ until we get to a non-diamond formula. During the unfolding of diamond formulas we argue the only way for the unfolded formula not to apply is for the subformula, after the diamond modality, not to apply.

The encoding of $\langle-(\alpha,\gamma)\rangle.\mathcal{F}$ holds if there is a transition with action different than α_π after which the encoding of $\mathcal{F}$ holds, or if there is at least one transition with action α_π but γ_π does not hold and the encoding of $\mathcal{F}$ does. α_π is the MMC action corresponding to kell-m action α, and γ_π is the MMC kell containment condition corresponding to kell-m's γ. Notice the encoding implements the semantics of $\langle-(\alpha,\gamma)\rangle.\mathcal{F}$ as defined in Figure 14.

$$
\begin{array}{rcl}
\mathcal{T}_p(\mathtt{tt}) & \stackrel{def}{=} & \mathtt{tt} \\
\mathcal{T}_p(\mathtt{ff}) & \stackrel{def}{=} & \mathtt{ff} \\
\mathcal{T}_p(\neg\mathcal{F}) & \stackrel{def}{=} & \mathtt{neg\_form}(F), \text{ with } F \stackrel{def}{=} \mathcal{T}_p(\mathcal{F}) \\
\mathcal{T}_p(\mathcal{C}.\mathcal{F}) & \stackrel{def}{=} & \mathtt{pred}(\mathcal{C}, \mathcal{T}_p(\mathcal{F})) \\
\mathcal{T}_p(\mathcal{F}_1 \wedge \mathcal{F}_2) & \stackrel{def}{=} & \mathcal{T}_p(\mathcal{F}_1) \wedge \mathcal{T}_p(\mathcal{F}_2) \\
\mathcal{T}_p(\mathcal{F}_1 \vee \mathcal{F}_2) & \stackrel{def}{=} & \mathcal{T}_p(\mathcal{F}_1) \vee \mathcal{T}_p(\mathcal{F}_2) \\
\mathcal{T}_p(\langle\delta\rangle.\mathcal{F}) & \stackrel{def}{=} & \langle\alpha_\pi\rangle.(\gamma_\pi \wedge \mathcal{T}_p(\mathcal{F})), \text{ with } \mathcal{AE}(\delta) = (\alpha_\pi, \gamma_\pi) \\
\mathcal{T}_p(\langle -\delta\rangle.\mathcal{F}) & \stackrel{def}{=} & \langle -\alpha_\pi\rangle.\mathcal{T}_p(\mathcal{F}) \vee \langle\alpha_\pi\rangle.(\mathcal{T}_p(\mathcal{F}) \wedge \neg\gamma_\pi), \text{ with } \mathcal{AE}(\delta) = (\alpha_\pi, \gamma_\pi) \\
\mathcal{T}_p(\langle S\rangle.\mathcal{F}) & \stackrel{def}{=} & \left\{ \begin{array}{l} \mathtt{ff}, \text{ if } S = \emptyset; \text{ otherwise:} \\ \mathcal{T}_p(\langle\delta_1\rangle.\mathcal{T}_p(\mathcal{F})) \vee \mathcal{T}_p(\langle\delta_2\rangle.\mathcal{T}_p(\mathcal{F})) \vee \cdots \vee \mathcal{T}_p(\langle\delta_n\rangle.\mathcal{T}_p(\mathcal{F})), \\ \text{with } S = \{\delta_1, \delta_2, ..., \delta_n\} \end{array} \right. \\
\mathcal{T}_p(\langle \text{-S}\rangle.\mathcal{F}) & \stackrel{def}{=} & \left\{ \begin{array}{l} \langle -\rangle.\mathcal{T}_p(\mathcal{F}), \text{ if } S = \emptyset; \text{ otherwise:} \\ \mathcal{T}_p(\langle -\delta_1\rangle.\mathcal{T}_p(\mathcal{F})) \wedge \mathcal{T}_p(\langle -\delta_2\rangle.\mathcal{T}_p(\mathcal{F})) \wedge \cdots \wedge \mathcal{T}_p(\langle -\delta_n\rangle.\mathcal{T}_p(\mathcal{F})), \\ \text{with } S = \{\delta_1, \delta_2, ..., \delta_n\} \end{array} \right. \\
\mathcal{T}_p([\delta].\mathcal{F}) & \stackrel{def}{=} & \mathtt{neg\_form}(F), \text{ with } F \stackrel{def}{=} \langle\alpha_\pi\rangle.(\gamma_\pi \wedge \mathtt{neg\_form}(F')), \\
 & & F' \stackrel{def}{=} \mathcal{T}_p(\mathcal{F}), \mathcal{AE}(\delta) = (\alpha_\pi, \gamma_\pi) \\
\mathcal{T}_p([-\delta].\mathcal{F}) & \stackrel{def}{=} & \mathcal{T}_p([-\{\delta\}].\mathcal{F}) \\
\mathcal{T}_p([S].\mathcal{F}) & \stackrel{def}{=} & \left\{ \begin{array}{l} \mathtt{tt}, \text{ if } S = \emptyset; \text{ otherwise:} \\ \mathcal{T}_p([\delta_1].\mathcal{T}_p(\mathcal{F})) \wedge \mathcal{T}_p([\delta_2].\mathcal{T}_p(\mathcal{F})) \wedge \cdots \wedge \mathcal{T}_p([\delta_n].\mathcal{T}_p(\mathcal{F})), \\ \text{with } S = \{\delta_1, \delta_2, ..., \delta_n\} \end{array} \right. \\
\mathcal{T}_p([\text{-S}].\mathcal{F}) & \stackrel{def}{=} & \left\{ \begin{array}{l} [-].\mathcal{T}_p(\mathcal{F}), \text{ if } S = \emptyset; \text{ otherwise:} \\ \mathcal{T}_p([-\delta_1].\mathcal{T}_p(\mathcal{F})) \wedge \mathcal{T}_p([-\delta_2].\mathcal{T}_p(\mathcal{F})) \wedge \cdots \wedge \mathcal{T}_p([-\delta_n].\mathcal{T}_p(\mathcal{F})), \\ \text{with } S = \{\delta_1, \delta_2, ..., \delta_n\} \end{array} \right. \\
\mathcal{T}_p(F(\widetilde{p})) & \stackrel{def}{=} & \mathtt{form}(F'), \text{ with } F' \stackrel{def}{=} \mathcal{T}_p(\mathcal{F}_d\{\widetilde{p}/\widetilde{c}\}), \text{ having } F(\widetilde{c}) \stackrel{def}{=} \mathcal{F}_d
\end{array}
$$

Where,

$$
\mathcal{AE}_p(\delta) \stackrel{def}{=} \left\{ \begin{array}{l}
(\mathcal{A}_p(\alpha), \mathcal{E}_p(\gamma, K_c)), \text{ if } \delta \equiv (\alpha, \gamma) \text{ and } \left\{ \begin{array}{l} \mathcal{A}_p(\alpha) = \overline{a}(\widetilde{w}, K_a, K_c, B) \\ \text{or} \\ \mathcal{A}_p(\alpha) = \overline{k}(h, K_a, K_c, B) \end{array} \right. \\
(\mathcal{A}_p(\alpha), \mathcal{E}_p(\gamma, K_a)), \text{ if } \delta \equiv (\alpha, \gamma) \text{ and } \left\{ \begin{array}{l} \mathcal{A}_p(\alpha) = a(\widetilde{c}, K_a, K_c, B) \\ \text{or} \\ \mathcal{A}_p(\alpha) = k(x, K_a, K_c, B) \end{array} \right. \\
(\mathcal{A}_p(\alpha_\tau), \mathcal{E}_p(\gamma_a, K_a) \wedge \mathcal{E}_p(\gamma_c, K_c)), \\
\qquad \text{if } \delta \equiv (\alpha_\tau, \gamma_a, \gamma_c) \text{ and } \left\{ \begin{array}{l} \mathcal{A}_p(\alpha_\tau) = \tau(\overline{a}(\widetilde{w}, K_a, K_c, B)) \\ \text{or} \\ \mathcal{A}_p(\alpha_\tau) = \tau(\overline{k}(h, K_a, K_c, B)) \end{array} \right.
\end{array} \right.
$$

Figure 27: Encoding of kμ Formulas in MMC

We use a well known modality equivalence for the encoding of $[\delta].\mathcal{F}$ [5]: $[\delta].\mathcal{F} \equiv \neg(\langle\delta\rangle.\neg\mathcal{F})$. The encoding of the other formulas follow from Figure 14.

7 Conclusions

In this report we presented kell-m, a higher-order, asynchronous process algebra with hierarchical localities. Systems are represented in kell-m as processes executing in parallel. Processes communicate via channels and processes can be located within kells. Kells themselves can be located within other kells forming a kell containment hierarchy. Both channels and kells are identified by name. Names and processes can be transmitted as part of a channel communication.

Two kinds of process concretions are supported in the algebra: writes on channels ($\overline{a}(\widetilde{w})$), and executing kells ($K[P]$). The corresponding abstractions are channel patterns ($a(\widetilde{c})$) and kell patterns ($K[x]$) in triggers ($\xi \triangleright P$). When a channel concretion and channel abstraction match, a channel communication occurs, and the values written to the channel are received by the trigger where the abstraction is specified. When a kell concretion and kell abstraction are matched, the kell is passivated, and its process is available, via a variable, to the trigger where the kell abstraction is specified.

We presented the operational semantics for kell-m. These semantics specify how processes evolve as they communicate and are passivated. We defined two LTS and one reduction semantics, and used them to specify the behavioural equivalences for kell-m.

In this report we also presented kμ, a modal temporal logic formalism for specifying conditions on the LTSs obtained using the extended kell-m semantics presented in Section 3.4. kμ is an extension of the $\pi\mu$-calculus, a logic with π-calculus modalities.

We also presented and encoding of kell-m processes as MMCπ processes. MMCπ is the process algebra implemented by the Mobility Model Checker (MMC). A callback feature in MMC allows us to deal with higher-order expressions in kell-m. kμ formulas are encoded using MMC's formalisms for property specification.

References

[1] Mobility Model Checker. `http://www.cs.sunysb.edu/~{}lmc/mmc/`, 2003. The Logic-Based Model Checking Project. 1, 18

[2] The XSB Logic Programming System, Version 3.2 (Kopi Lewak). `http://xsb.sourceforge.net/`, March 2009. XSB Research Group. 18

[3] P. Bidinger, A. Schmitt, and J.-B. Stefani. An Abstract Machine for the Kell Calculus. In M. Steffen and G. Zavattaro, editors, *7th IFIP Internaional Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, volume 3535 of *Lecture Notes in Computer Science*, pages 31–46. Lecture Notes in Computer Science, Springer-Verlag, 2005. 1, 3

[4] W. Chen and D. S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, 1996. 18

[5] M. Dam. Proof Systems for Pi-Calculus Logics. In R. J. de Queiroz, editor, *Logic for Concurrency and Synchronisation*, pages 145–212. Kluwer Academic Publishers, 2003. Trends in Logic – Studia Logica Library. 42

[6] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Lecture Notes in Computer Science, Springer-Verlag, 1980. 23

[7] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, Computer Laboratory, University of Cambridge, 1999. 4, 5

[8] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Part I. Technical Report ECS-LFCS-89-85, Computer Science Department, University of Edinburgh, 1989. 18

[9] R. Milner and D. Sangiorgi. Barbed Bisimulation. In *ICALP '92: Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, pages 685–695, London, UK, 1992. Springer-Verlag. 10

[10] R. M. Needham. Names. In S. Mullender, editor, *Distributed Systems*, pages 89–101. ACM Press, 1989. 1

[11] J. Parrow. An Introduction to the Pi-Calculus. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001. 4, 9

[12] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, Scotland, UK, 1992. 9, 19

[13] D. Sangiorgi. From pi-calculus to Higher-Order pi-calculus - and Back. In *TAPSOFT '93: Proceedings of the International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, pages 151–166, London, UK, 1993. Springer-Verlag. 19

[14] D. Sangiorgi. Bisimulation: From the Origins to Today. In H. Ganzinger, editor, *Proceedings of the Nineteenth Annual IEEE Symp. on Logic in Computer Science, LICS 2004*, pages 298–302. IEEE Computer Society Press, July 2004. Invited Talk. 11

[15] A. Schmitt and J.-B. Stefani. The Kell Calculus: A Family of Higher-Order Distributed Process Calculi. In C. Priami and P. Quaglia, editors, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 146–178. Springer-Verlag, 2004. 1, 3

[16] J.-B. Stefani. A Calculus of Kells. In *In Proceedings 2nd International Workshop on Foundations of Global Computing*, volume 85. Electronic Notes in Theoretical Computer Science, Elsevier, 2003. 1, 3

[17] P. Yang, C. R. Ramakrishnan, and S. A. Smolka. A Logical Encoding of the Pi-Calculus: Model Checking Mobile Processes Using Tabled Resolution. In *VMCAI 2003: Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 116–131, London, UK, 2003. Springer-Verlag. 1, 14, 18, 38