A Calculus of Higher-Order Distributed Components

Jean-Bernard Stefani

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Calculus of Higher-Order Distributed Components

Jean-Bernard Stefani

N° 4692

January 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4692--FR+ENG

A Calculus of Higher-Order Distributed Components

Jean-Bernard Stefani

Thème 1 — Réseaux et systèmes
Projets SARDES

Rapport de recherche n° 4692 — January 2003 — 53 pages

Abstract: This report presents a calculus for higher-order distributed components, the Kell calculus. The calculus can be understood as a direct extension of the higher-order π-calculus with programmable locations. The report illustrates the expressive power of the Kell calculus by encoding several process calculi with explicit locations, including Mobile Ambients, the Distributed Join calculus and the M-calculus. The latter encoding demonstrates that the Kell calculus retains the expressive power of the M-calculus but in a much simpler setting.

Key-words: process calculus, distributed programming, programming model, mobile processes, π-calculus, ambients, concurrent objects.

This research has been supported in part by the IST Global Computing project Mikado (IST-2001-32222).

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Un calcul d'ordre supérieur pour composants répartis

Résumé : Ce rapport de recherche présente un nouveau calcul de processus réparti d'ordre supérieur, le Calcul de Cellules. Ce nouveau calcul peut être compris comme une extension directe du π-calcul d'ordre supérieur avec des localités programmables. Le rapport illustre le pouvoir expressif du Calcul de Cellules par le codage de plusieurs calcul de processus avec localités explicites tels que Mobile Ambients, le Join calcul réparti et le M-calcul. Le codage de ce dernier montre que l'on peut conserver le pouvoir expressif du M-calcul dans un cadre beaucoup plus simple.

Mots-clés : calcul de processus, modèle de programmation, programmation répartie, processus mobiles, π-calcul, ambients, objets concurrents.

Contents

1 Introduction

Several distributed process calculi have been proposed in the last decade. We can roughly classify them in three classes:

- Ambient calculi, such as the original Mobile Ambients calculus [7, 6] and the subsequent variants such as Safe Ambients [17], Safe Ambients with Passwords [20], Boxed Ambients [5], Controlled Ambients [26].
- Higher-order process calculi such as as Facile/CHOCS [16, 27] and D$\pi\lambda$ [30], that model process mobility via higher-order communication and remote process execution.
- Variants of the first-order asynchronous π-calculus with explicit localities such as the Distributed Join calculus [11, 10, 9], Nomadic Pict [29], DiTyCo [18], or the π_{1l}-calculus [2], that feature process migration primitives (`go` in the Distributed Join calculus, `spawn` in the π_{1l}-calculus, `migrate` in Nomadic Pict).

These different calculi suffer from various limitations, which we discuss below. A recent proposal, the M-calculus [25], combines features from calculi in these different groups, but also suffers from several infelicities.

1.1 Limitations of ambient calculi

Ambient calculi provide a simple model of hierarchical locations with fine-grained control over location moves and communications. This is expecially true in the variants with co-capabilities (e.g. Safe Ambients, Safe Ambients with passwords and Boxed Ambients). Unfortunately, these calculi suffer from two broad limitations: (1) a difficult implementation in a distributed setting, and (2) no support for different location semantics.

Difficult distributed implementation. The basic mobility primitives of Ambient calculi (the `in` and `out` capabilities) require an atomic rendez-vous between at least 2 ambients. Consider the rule that governs the `in` primitive, for instance:

$$a[\texttt{in}\, b.P \mid Q] \mid b[R] \rightarrow b[R \mid a[P \mid Q]]$$

This rule requires, in one atomic step, to move ambient a to ambient b and to release the continuation P in ambient a. Even if one disregards the atomicity requirement on the continuation, such atomic transitions are costly to implement in a distributed setting, where ambients may be located on different machines. Indeed, moving ambient a to ambient b requires that ambient b be located and locked in place for the duration of the move from a to b. The high cost incurred by the implementation of such atomic transitions is clearly demonstrated by the implementation of Mobile Ambients in the Distributed Join calculus[12]. This implementation suggests that taking Mobile Ambients as a basis for distributed computation would be inefficient, since a large number of useful applications only require simple non-transactional asynchronous communication.

Recent work on the PAN distributed abstract machine for Safe Ambients [23], reinforces this point. The distributed implementation of ambients proposed there does away with the problem by

implementing the `in` and `out` capabilities locally (relying on co-capabilities and single-threadedness), and interpreting the `open` capability as a move to the implicit location of the parent ambient. In this interpretation, ambients do not correspond to a physical distribution of a computation, but merely to a logical one. Furthermore, work on Boxed Ambients successfully argues against the `open` capability, which means that the PAN distributed interpretation of ambients is questionable.

Actually the only meaningful distributed interpretation of ambients we can think of (i.e. where ambients can be mapped to different machines in a computer network), would consist in applying a sorting that would distinguish between site ambients, i.e. ambients that correspond to physical sites, and movable ambients, i.e. ambients that can be sent between sites. The Mobile Ambients primitives thus would apply between two movable ambients or between a movable ambient and a site ambient, but not between two site ambients (or at an understood higher cost than movable ambients). In this interpretation, we recover asynchrony since communication between sites now corresponds to the following sequence: an `out` to get out of the source site ambient, followed by an `in` to get into the destination site ambient (the root ambient plays the role of the network that connects the sites). This interpretation is strikingly similar to the interpretation of localities in the Distributed Join calculus implementation: top-level localities are mapped onto sites (machines) which communicate asynchronously. If one follows this interpretation, it becomes clear that movable ambients should be interpreted as higher-order messages that can be exchanged between site ambients and, locally within a site, between other movable ambients. One would thus be led to a model of localities where such possibility would be taken as primitive, since it corresponds directly to the basic capabilities of asynchronous networks and systems. This is actually the path that the M-calculus took and which we follow again with the Kell calculus.

No support for different location semantics. Another limitation of Ambient calculi is their reliance on a fixed semantics for ambients, which changes from calculus to calculus, depending on the particular phenomena which each calculus intends to capture. This, in our view, partly explains the recent multiplication of Ambient calculi. One should instead consider means, within a single calculus, to express the different interaction protocols which may characterize different forms of locations. These protocols could be quite complex. Consider for instance interaction protocols concerned with access control. There are several mechanisms that can be used for controlling access to given resources, and an even wider range of policies for governing the use of these mechanisms. Furthermore, in an open environment, it seems clear that one cannot avoid the recourse to run-time checks and dynamic structures, if only to take into account the need for dynamic modifications of access control policies. It is therefore difficult to envisage determining once and for all a precise set of primitives for enforcing access control to distributed locations. Instead, one should expect from a distributed calculus the ability to *define* the required access control behavior, much as any other type of distributed behavior.

1.2 Limitations of higher-order process calculi

Distributed higher-order process calculi have asynchronous higher-order communication as a primitive, but they have two broad limitations: (1) they lack an explicit notion of location, and (2) they do not allow a running process to be interrupted.

No explicit notion of location. The lack of an explicit notion of location prevents these calculi to account for potential failures or to provide a basis for access control. Such aspects can be taken into account at a semantical level, when trying to account for the operational semantics of a distributed implementation, as is the case e.g. in the CHAM semantics of Facile described in [16]. However, as with the Distributed Join calculus discussed below, there is a need to support different locality semantics, corresponding to different failure modes or different access capabilities. The lack of an explicit concept of locality in such calculi hampers such developments. One could think of encoding such concepts in these calculi: for instance, access control could be put in place by means of wrappers around protected processes, and failures could be modelled by means of interruptible processes. However, this begs the question of knowing what primitives are required to emulate in these encodings. Notice, in particular, that the definition of interruptible processes is not trivial to define in an asynchronous concurrency model. This in turn suggests that enhancing such calculi with appropriate constructs for process interruption would be the easiest path to follow[1].

No interruptible processes. The second broad limitation of such calculi lies in the fact that they do not allow for interrutible process. Not only does that hamper the development of failure semantics, but it prevents a running process to be migrated to a different locality, unless the process has been explicitly defined to allow for such a migration. This is a severe limitation for modelling or programming reconfigurable systems, i.e. component-based systems where components can be installed, moved or replaced dynamically. While dynamic reconfiguration capabilities can certainly be programmed in these calculi, such modelling can become quite intricate, and in any case it still begs the question of knowing exactly what primitive constructs are required to support reconfiguration. In this case, it seems that the ability to identify and manipulate processes as data elements would be a crucial requirement.

1.3 Limitations of the DJoin calculus

Variants of the first-order asynchronous π-calculus with localities suffer in turn from several limitations, typically insufficient control over process mobility and communication, and lack of support for different location semantics. As an illustration, let us discuss in more detail the limitations of the Distributed Join calculus[2].

The Distributed Join (DJoin) calculus constitutes an important milestone towards the definition of an effective programing model for distributed and mobile computation. Its asynchronous character and its resource locality property (receptors are uniquely defined and located), reflect the basic structure of communication in today's point-to-point asynchronous internetworks, and make a distributed implementation of the calculus effective. Hierarchical locations in the DJoin calculus allows strong mobility of DJoin processes, and can model (fail-stop) failures - a crucial requirement for practical distributed programming. The DJoin calculus, however, has several limitations: (1) it offers insuf-

[1] In a different context, one can note that the disruption operator in the specification language LOTOS was introduced precisely to allow the modelling of interruptions and failures in an otherwise asynchronous concurrency setting.

[2] Not all the calculi in this group suffer from the exact same limitations as the DJoin calculus. For instance, Nomadic Pict and the π_{1l} calculus do support a form a dynamic binding, even if they lack programmable locations and offer insufficient control over mobility and communication. However, we think the DJoin calculus is one of the most interesting calculi in this class, and certainly a practical distributed programming model ought to be at least as efficiently implementable as the DJoin.

ficient control over process mobility; (2) if offers insufficient control over communications; (3) it does not support dynamic binding; (4) it does not support the definition of locations with different semantics.

Insufficient control over process mobility. In the DJoin calculus, it is not possible to to prevent a location from migrating to another location. Thus, except in presence of failures, the DJoin command $\mathbf{go}\langle a, \kappa\langle\rangle\rangle$ which instructs the current location to move to location a, always succeeds and cannot be blocked by the receiving location a. This means that it is always possible to move a running process to a specified location, potentially raising security concerns since this prevents, for instance, access control checks to occur prior to migrating an active agent (implemented as a DJoin location). As a minimum, one would require the functional equivalent of Safe Ambients co-capabilities [17], to effect migration only when the receiving location allows it.

Insufficient control over communications. Once a resource (a DJoin calculus definition) has been activated, and its channels have been communicated, it is difficult to prevent access to that resource and to erect the equivalent of a firewall as can be done e.g. in Mobile Ambients [7]. To enforce more control over communications, one could think of making use of forwarders not dissimilar to the relay processes employed in [9] for ensuring a fully abstract simulation of the π-calculus in the Join calculus. However this use would be cumbersome and should be systematically enforced (e.g. through the use of appropriate syntactic extensions to the calculus), which begs the question of defining a proper control construct in the first place.

No support for dynamic binding. By dynamic binding, we refer to the possibility of binding, at run-time, a channel name in a process to a particular resource, depending on the location of said process. This feature is important for an effective distributed programming model. In a distributed setting, one can expect for instance to have multiple copies (or variants) of the same service at different sites, if only for cost or security reasons (sending a remote message can be more expensive or less secure than a local one). A mobile agent should thus be able to access the local copy of a service without necessarily having to learn about the local names of the service. Of course, one could think of using a name service and asociated name translation mechanisms (which can be encoded in the DJoin calculus) to implement the necessary indirection between service names and actual channel names delivering the service, but note that this begs the question of how one can bind to the local copy of the naming service to begin with, unless one assumes a unique centralized server. Also, one could argue that dynamic binding appears to be more primitive than this encoding in the DJoin calculus would imply. Bootstrapping, for instance, usually relies, not on a central repository, but on well-known names, which are dynamically bound to the (local) objects of interest (memory locations, name services, etc).

No support for different location semantics. Locations in the DJoin calculus are defined to be fail-stop. It would be possible to modify the RCHAM semantics of the calculus to accomodate different failure models (e.g. omission failures, Byzantine failures). However a more satisfactory answer would provide the means, *within the calculus itself, to define the expected or needed location behaviour*. This reinforces the first two points above: it should be possible, in the calculus, to define the exact behavior one expects from a location, whether for modelling particular failure modes, or for supporting different forms of access control.

1.4 Limitations of the M-calculus

A first attempt to remedy the shortcomings of the DJoin calculus while preserving its key properties (hierarchical locations, transparent routing, implementability) is the M-calculus [25]. In the M-calculus, programmable locations provide a means to retain the transparent routing of the DJoin, while offering a high degree of control over communications in and out of a location. The key idea behind programmable locations is that each location can be endowed with a controller process that acts as a filter of messages. Since the calculus is higher-order, migration is merely communication of messages that carry *thunks*, i.e. frozen processes. Therefore controlling migration in the M-calculus is the same as controlling communications.

While the M-calculus manages to remedy the above shortcomings of the DJoin calculus, it is plagued by a few infelicities: (1) multiple routing rules, (2) complex passivation construct, and (3) complex combination of functional, concurrent and distributed constructs.

Multiple routing rules. Because of the structure of programmable locations, the operational semantics of the calculus contains several routing rules that take care of the different ways a message can be handled when entering or leaving a location. These rules are variants of one another, and involve the use of special channels for the interception of messages by controllers. Also, the form of the rules clearly make apparent that routing in the M-calculus deal with multiple boundary crossings: between environment and controller (both ways), and between controller and content of a location (both ways). Clearly, one would like to find a simpler, more abstract scheme to account for these different rules and situations.

Complex passivation construct. The M-calculus contains a location passivation primitive whose semantics is a bit involved and which would probably benefit from being split into simpler constructs. In particular, passivation plays two roles: to interrupt a running location, and to reify the contents of a location in thunks for further handling.

Complex combination of functional, concurrent and distributed constructs. The M-calculus involves a combination of Join calculus and λ-calculus constructs which adds to the calculus complexity. The λ-calculus application and abstraction were added to the M-calculus to allow a continuation-free style similar to the Blue calculus [4], and to form the passivation primitive. While this combination provides a taste of what the hybridation of a functional programming language with distributed programming features would look like, it may obscure the fundamental primitives for distributed computation.

1.5 Introducing the Kell calculus

We introduce in this paper a new process calculus, called the Kell calculus. The calculus is intended to overcome the limitations of the previous calculi discussed above.

The basic constructs of the Kell calculus are similar to those of the π-calculus [22]. It contains the null process ($\mathbf{0}$), names (a), variables (x), restriction on names ($\nu a.$), parallel composition of processes ($P \mid Q$), and receivers, which we also call triggers ($\xi \triangleright P$). To these constructs, we add a single notion, that of kell, which takes two forms, a passive one ($a \circ P$), and an active one ($a \bullet P$).

Triggers are close to π-calculus receivers and to Join calculus definitions. In trigger $\xi \triangleright P$, ξ is called the trigger pattern and corresponds to the join pattern in a Join calculus definition. The

Kell calculus is actually parameterized by the language used to define trigger patterns, i.e. the Kell calculus can be understood as a family of calculi that share a common core (described in Section 2 below), but vary on the language of patterns used. In contrast to Join calculus definitions, a trigger disappears after it has been triggered. Because the Kell calculus is higher-order, receptive triggers, similar to Join calculus definitions, can in fact be defined. We note $\xi \diamond P$ receptive triggers.

Active kells are named processes of the form $a \bullet P$. Passive kells are named processes of the form $a \circ P$. Operators $\circ$ and $\bullet$ are control operators in their own right. A construct of the form $a \circ P$ freezes the execution of process P, which can be released though the a handle. In other words $a \circ \cdot$ is a form of box[3] that prevents the execution of processes inside it, and that can be opened through the a handle. Likewise, $a \bullet \cdot$ is a box that allows the execution of processes inside it, and that can be manipulated through the a handle. Coupled with communications, these two operators can be used, for instance, to implement control operations on processes.

Kells play the role of both locations and messages in the Kell calculus. A kell $a \circ P$ or $a \bullet P$ can be interpreted as a message whose destination is a and whose payload is P. A construct like $a \circ (a_1 \circ P_1 \mid \ldots \mid a_n \circ P_n)$ can be seen as a message whose destination in a and which has n arguments named $a_1, \ldots, a_n$. Such a construct is also just like a (named) record. A kell $a \bullet P$ can be interpreted as an active location of name a: process P can evolve freely inside $a \bullet P$ and receive messages (i.e. other kells) from the environment of $a \bullet P$.

In contrast to the Join calculus, we do not impose syntactic conditions on the unicity of receivers, i.e. several triggers that match the same set of messages may occur in different kells in the same execution context. This is essential to model dynamic binding features. For this reason, we avoid the static scoping of the DJoin and use the π-calculus restriction. Ensuring the determinacy of communication and message routing can be done as in the M-calculus by means of a type system once the pattern language is fixed.

The intuition behind the notion of kell is that a kell corresponds to a component, with data and behavior parts held by subordinate kells and triggers, that can serve as a wrapper or firewall for other (subordinate) components. Thus, kells of the form $a \bullet (a_1 \circ P_1 \mid \ldots \mid a_n \circ P_n \mid \xi_1 \triangleright Q_1 \mid \ldots \mid \xi_l \triangleright Q_l)$, resemble objects, with name a, private data elements $a_i \circ P_i$, and methods $\xi_j \triangleright P_j$.

Kells can move in and out of other kells, possibly carrying active elements and thus, active processes. A kell $M = a \bullet Q$ can only enter another kell b if there is a trigger $\xi \triangleright P$ inside the kell whose pattern matches $a \bullet P$ (the term $P\theta$ below is the result of the trigger reaction to the receipt of message M):

$$M \mid b \bullet (\xi \triangleright P \mid \ldots) \rightarrow b \bullet (P\theta \mid \ldots)$$

Likewise, a kell M can only leave a kell b if there is a trigger outside the kell whose pattern matches the message:

$$\xi \triangleright P \mid b \bullet (M \mid \ldots) \rightarrow P\theta \mid b \bullet (\ldots)$$

A kell thus has full control over its communications with its environment and communication proceeds in a local manner, by crossing one kell boundary $b \bullet (\cdot)$ at a time. In this respect, the Kell calculus resembles the Seal calculus [28] and Boxed Ambients [5]. Indeed, one way to understand the Kell calculus is as an asynchronous form of Boxed Ambients where mobility primitives have

[3] One can draw an analogy here with the **box** operator of the PIC action calculus [21].

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid a \mid x \mid \xi \triangleright P \mid \nu a.P \mid (P \mid P) \mid a * P \\ * & ::= & \bullet \mid \circ \end{array}$$

Figure 1: Syntax of the Kell Calculus

been replaced by higher-order communications. Thus, a kell can communicate only with its enclosing parent (the super-kell) and the kells it encloses (sub-kells). Any form of communication with remote kells, i.e. kells not immediately above or below the current kell in the kell forest, must be mediated by the super-kell or sub-kells.

To preserve the benefits of transparent routing as in the DJoin, we retain the solution adopted in the M-calculus, namely to provide for an easy encoding of transparent message routers. This in turn relies on the definition of pattern languages that are able to introspect the contents of a given kell, in order for a kell to decide whether a given trigger pattern is local to the kell or not. As will be shown below, this allows us to encode the M-calculus in the Kell calculus, thus retaining the full expressive power of the former, but in a much simpler setting.

1.6 Organization of the paper

The paper is organized as follows. Section 2 defines the syntax and operational semantics of the Kell calculus. The operational semantics is given in two forms: a reduction semantics and a labelled transition system semantics, which are shown to coincide. Section 3 presents various examples in the Kell calculus equipped with a simple pattern language, together with encodings of different calculi such as the λ-calculus, the π-calculus, Mobile Ambients, and a concurrent object-based calculus. Section 4 introduces a more sophisticated pattern language and presents the encoding of the Distributed Join calculus and of the M-calculus in the resulting Kell calculus. Section 5 contains additional discussion of the Kell calculus and its subcalculi. Section 6 concludes the paper with some discussion of future research directions.

2 The Kell Calculus

2.1 Syntax

The syntax of the Kell calculus is given in Figure 1. It is parameterized by the pattern language used to define patterns ξ in triggers $\xi \triangleright P$. We assume an infinite set of *names*, noted N, and an infinite set of *variables*, noted V, such that $\mathsf{N} \cap \mathsf{V} = \emptyset$. We let a, b, c, d and their decorated variants range over N, and we let x, y, p, q range over V. The set of *identifiers*, L, is defined as $\mathsf{L} = \mathsf{N} \cup \mathsf{V}$.

Terms in the Kell calculus grammar are called *processes*. We note K the set of Kell calculus processes. We let P, Q, R, and their decorated variants range over processes. We call *kell* a process of the form $a * P$. The name a in a kell $a * P$ is called the name of the kell. Kells of the form $a \bullet P$ are called active kells. Kells of the form $a \circ P$ are called passive kells. We call *message* a Kell calculus process that is a process that is either a name or a kell. We let M, N and their decorated variants

$$\begin{array}{lcl} \mathbf{C} & ::= & \cdot \mid P \mid \xi \triangleright \mathbf{C} \mid \nu a.\mathbf{C} \mid (P \mid \mathbf{C}) \mid a * \mathbf{C} \\ \mathbf{E} & ::= & \cdot \mid a \bullet \mathbf{E} \mid \nu a.\mathbf{E} \mid P \mid \mathbf{E} \end{array}$$

Figure 2: Syntax of Contexts

range over messages and parallel composition of messages. Parallel composition of messages are therefore defined by the following syntax:

$$M, N ::= \mathbf{0} \mid a \mid a * P \mid (M \mid N)$$

In a kell of the form $a \bullet(\ldots \mid a_j * P_j \mid \ldots \mid Q_k \mid \ldots)$ we call *subkells* the processes $a_j * P_j$.

In a term $\nu n.P$, the scope extends as far to the right as possible. We use $\widetilde{P}$ to denote finite vectors of terms $(P_1, \ldots, P_q)$. We use standard abbreviations from the the π-calculus: $\nu a_1 \ldots a_q.P$ for $\nu a_1 \ldots . \nu a_q.P$, or $\nu \widetilde{a}.P$ if $\widetilde{a} = (a_1 \ldots a_q)$. By convention, if the name vector $\widetilde{a}$ is null, then $\nu \widetilde{a}.P = P$. We abbreviate $a\langle P_1, \ldots, P_n \rangle$ a kell of the form $a \circ (1 \circ P_1 \mid \ldots \mid n \circ P_n)$, where we consider that $1, \ldots, n, \ldots$ belong to N. We also note $\prod_{j \in J} P_j$, $J = \{1, \ldots, n\}$ the parallel composition $(P_1 \mid (\ldots (P_{n-1} \mid P_n) \ldots))$. By convention, if $J = \emptyset$, then $\prod_{j \in J} P_j = \mathbf{0}$.

A Kell calculus context is a term $\mathbf{C}$ built according to the same grammar than for standard Kell calculus terms, plus a constant $\cdot$, the hole. Filling the hole in $\mathbf{C}[\cdot]$ with a Kell calculus term Q results in a Kell calculus term noted $\mathbf{C}[Q]$. We let $\mathbf{C}$ and its decorated variants range over Kell calculus contexts. We make use of a specific form of contexts, called execution contexts (noted $\mathbf{E}$), which are used to specify the operational semantics of the calculus. The syntax of *contexts* is given in Figure 2.

A pattern acts as a binder in the calculus. A pattern can bind *name markers*, of the form (a), where $a \in \mathsf{N}$, and *process markers*, of the form (x), where $x \in \mathsf{V}$. All markers appearing in a pattern ξ are bound by the pattern. Name markers can only match names. Process markers can match any process, including processes reduced to a name. In a slight abuse of notation, we frequently dispense with the parenthesis $(.)$ around markers (especially process markers) when it is clear from the context which names act as markers[4].

A process P matches a pattern ξ if there is a substitution (i.e. a function from names to Kell calculus terms that is the identity except on a finite set of names), θ, that maps identifiers u appearing as markers in ξ (i.e. $u \in \mathsf{bv}(\xi)$) on Kell calculus terms, such that $\xi\theta = P$ (i.e. such that the image of pattern ξ under substitution θ is the process P). In the Kell calculus, we also make use of context-dependent patterns. Such patterns typically include a side condition or a guard that depends on the current evaluation context. Matching for these patterns is defined as for standard patterns, but using the notion of a context-dependent substitution. A context-dependent substitution θ is a function that maps pairs $\langle \mathbf{C}, u \rangle$ of Kell calculus contexts and identifiers onto Kell calculus terms. We note $P\theta_{\mathbf{C}}$ the image of the term P under substitution θ, given a context $\mathbf{C}$.

[4]Pattern languages used in this paper do not make use of free variables in trigger patterns. As a consequence, this convention can be employed systematically for process markers since there is no risk of confusing a process marker with a free variable.

$$
\begin{array}{ll}
\mathtt{fn}(\mathbf{0}) = \emptyset & \mathtt{fv}(\mathbf{0}) = \emptyset \\
\mathtt{fn}(a) = \{a\} & \mathtt{fv}(a) = \emptyset \\
\mathtt{fn}(x) = \emptyset & \mathtt{fv}(x) = \{x\} \\
\mathtt{fn}(\nu a.P) = \mathtt{fn}(P) \setminus \{a\} & \mathtt{fv}(\nu a.P) = \mathtt{fv}(P) \\
\mathtt{fn}(a * P) = \mathtt{fn}(P) \cup \{a\} & \mathtt{fv}(a * P) = \mathtt{fv}(P) \\
\mathtt{fn}(P \mid Q) = \mathtt{fn}(P) \cup \mathtt{fn}(Q) & \mathtt{fv}(P \mid Q) = \mathtt{fv}(P) \cup \mathtt{fv}(Q) \\
\mathtt{fn}(\xi \triangleright P) = \mathtt{fn}(\xi) \cup (\mathtt{fn}(P) \setminus \mathtt{bn}(\xi)) & \mathtt{fv}(\xi \triangleright P) = \mathtt{fv}(\xi) \cup (\mathtt{fv}(P) \setminus \mathtt{bv}(\xi))
\end{array}
$$

Figure 3: Free names and free variables

The other binder in the calculus is the ν. operator, which corresponds to the restriction operator of the π-calculus. Notions of free names ($\mathtt{fn}$) and free variables ($\mathtt{fv}$) are classical and are defined in Figure 3. We note $\mathtt{ln}(P)$ the set of names that occur in process P, and $\mathtt{bn}(P)$ the set of bound names of P ($\mathtt{ln}(P) = \mathtt{fn}(P) \cup \mathtt{bn}(P)$). We note $P =_\alpha Q$ when two terms P and Q are α-convertible.

We make the following assumptions on pattern languages:

- One can decide whether a pattern matches a given term and the result of applying a substitution on markers to a pattern ξ is a Kell calculus process. Generally, given a context $\mathbf{C}$, we say that a pattern ξ matches a kell calculus term P in context $\mathbf{C}$, if there exists a context-dependent substitution θ which maps names u appearing as markers in ξ to Kell calculus processes, such that $\xi\theta_{\mathbf{C}} = P$.
- A pattern language is compatible with the structural congruence defined below, i.e. if $P \equiv Q$ then there is no Kell calculus context that can distinguish between P and Q.
- Patterns can be sorted in at least two sorts S and H. The sort H corresponds to the sort of all patterns. The sort S corresponds to a subset of patterns that can match terms in contexts of the form $(\cdot \mid P)$. We note $\xi : \mathsf{T}$ to indicate that pattern ξ is of sort T.

2.2 Reduction Semantics

The operational semantics of the Kell calculus is defined in the CHAM style [3], via a structural equivalence (actually, a congruence) and a reduction relation. The structural congruence $\equiv$ is the smallest equivalence relation that verifies the rules in Figure 4 and the rules that state that the parallel operator $\mid$ is associative (rule S.PAR.ASSOC), commutative (rule S.PAR.COMM), and that $\mathbf{0}$ is a neutral element for the parallel operator (rule S.PAR.NIL). Note that, in rule S.TRIG, we rely on the existence of a structural equivalence relation on patterns, also noted $\equiv$.

The reduction relation $\rightarrow$ is the smallest relation satisfying the rules given in Figure 5. Notice that we allow pattern matching on active or passive elements within the same kell, and only on messages external to the kell, or within a subkell. This means that only messages are allowed to cross the boundary of a kell.

The basic reduction rules of the calculus R.RED.S and R.RED.H in Figure 5 look rather involved but their nature can be easily revealed by considering the derived rules IN and OUT in Figure 6. These rules correspond, respectively, to the case of messages entering a kell, and to the case of messages

$$\frac{a \notin fn(Q)}{(\nu a.P) \mid Q \equiv \nu a.P \mid Q}\ [\text{S.NU.PAR}] \qquad \frac{a \neq b}{\nu a.\nu b.P \equiv \nu b.\nu a.P}\ [\text{S.NU.COMM}]$$

$$\frac{a \neq b}{b \bullet \nu a.P \equiv \nu a.b \bullet P}\ [\text{S.NU.KELL}] \qquad \nu a.\mathbf{0} \equiv \mathbf{0}\ [\text{S.NIL}] \qquad \frac{\xi \equiv \zeta}{\xi \triangleright P \equiv \zeta \triangleright P}\ [\text{S.TRIG}]$$

$$\frac{P =_\alpha Q}{P \equiv Q}\ [\text{S.}\alpha] \qquad \frac{P \equiv Q}{\mathbf{C}[P] \equiv \mathbf{C}[Q]}\ [\text{S.CONTEXT}]$$

Figure 4: Structural congruence

$$\frac{\begin{array}{c}\xi : \mathsf{S} \\ \mathbf{C} = \cdot \mid N \mid Q \qquad Q = \prod_{j\in J} a_j \bullet (N_j \mid Q_j) \qquad Q' = \prod_{j\in J} a_j \bullet Q_j \qquad \xi\theta_{\mathbf{C}} \equiv N \mid \prod_{j\in J} N_j\end{array}}{\xi \triangleright P \mid N \mid Q \to P\theta_{\mathbf{C}} \mid Q'}\ [\text{R.RED.S}]$$

$$\frac{\begin{array}{c}\xi : \mathsf{H} \qquad \mathbf{C} = M \mid a \bullet (\cdot \mid Q \mid R) \\ Q = \prod_{j\in J} a_j \bullet (N_j \mid Q_j) \qquad Q' = \prod_{j\in J} a_j \bullet Q_j \qquad \xi\theta_{\mathbf{C}} \equiv M \mid N \mid \prod_{j\in J} N_j\end{array}}{a \bullet (\xi \triangleright P \mid N \mid Q \mid R) \mid M \to a \bullet (P\theta_{\mathbf{C}} \mid Q' \mid R)}\ [\text{R.RED.H}]$$

$$\frac{P \to Q}{\mathbf{E}[P] \to \mathbf{E}[Q]}\ [\text{R.CONTEXT}] \qquad \frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q}\ [\text{R.EQUIV}]$$

Figure 5: Reduction Relation

leaving a kell. These rules are themselves just extensions of the standard rule of (β) reduction with filtering: rule BETA in Figure 6. Rule IN and OUT indicate that messages can cross kell boundaries, and that crossing a kell boundary requires the presence of a trigger on the other side of the boundary. Note that rules IN, OUT, and BETA deal only with context-independent substitutions, i.e. they involve patterns that are not dependent on context. Note also that rules R.RED.S and R.RED.H take into account the possibility for patterns to match parallel composition of messages occurring at different levels (outside the receiving kell and within subkells of the receiving kell).

The rules R.RED.S and R.RED.H identify the form of contexts that can be used for pattern matching. This is another assumption we make on pattern languages: patterns of the sort S can only match terms in contexts that are of the form $\mathbf{E}[\cdot \mid M \mid Q]$, and patterns of the sort H, i.e. arbitrary patterns, can only match terms in contexts that are of the form $\mathbf{E}[M \mid a \bullet (\cdot \mid N \mid Q)]$, where Q is a Kell calculus process, and M, N are parallel compositions of messages. Intuitively, this means that context-dependent patterns can only introspect *locally*, i.e. on the contents of the immediate surrounding kell. More precisely, if $\xi : \mathsf{S}$, then $\xi\theta_{\mathbf{C}}$ is only defined for $\mathbf{C} \equiv \mathbf{E}[\mathbf{C}_s]$ and $\xi\theta_{\mathbf{C}} = \xi\theta_{\mathbf{C}_s}$, where $\mathbf{C}_s = \cdot \mid M \mid Q$. Likewise, if $\xi : \mathsf{H}$, then $\xi\theta_{\mathbf{C}}$ is only defined for

$$\frac{\xi\theta \equiv M}{a \bullet (\xi \triangleright P \mid Q) \mid M \to a \bullet (P\theta \mid Q)}\ [\textsc{In}] \qquad \frac{\xi\theta \equiv M}{\xi \triangleright P \mid a \bullet (M \mid Q) \to P\theta \mid a \bullet Q}\ [\textsc{Out}]$$

$$\frac{\xi\theta \equiv M}{\xi \triangleright P \mid M \to P\theta}\ [\textsc{Beta}]$$

Figure 6: Derived reduction rules I

$\mathbf{C} \equiv \mathbf{E}[\mathbf{C}_h]$ and $\xi\theta_{\mathbf{C}} = \xi\theta_{\mathbf{C}_h}$, where $\mathbf{C}_h = M \mid a \bullet (\cdot \mid N \mid Q)$. We make a further assumption on pattern matching, which is consistent with the previous assumption that a pattern language should be consistent with structural congruence, namely: for all $\mathbf{C}, \mathbf{C}', \xi, \xi', \theta$, if $\mathbf{C} \equiv \mathbf{C}'$ and $\xi \equiv \xi'$, then $\xi\theta_{\mathbf{C}} \equiv \xi'\theta_{\mathbf{C}'}$.

2.3 Labelled transition system semantics

We define in this section a labelled transition system for Kell calculus processes. Labels appearing in transitions take the form $\mathbf{K} : A$, where $\mathbf{K}$ is an action context and A is an action. Actions are defined by the following grammar, where $P \in \mathsf{K}$, and where $\tau, \epsilon \notin \mathsf{L}$:

$$\begin{array}{lcl} A & ::= & \epsilon \mid \tau \mid \alpha \mid \beta \mid \overline{\nu}a.A \mid \langle A \rangle \mid A \mid A \\ \alpha & ::= & a * P \\ \beta & ::= & (\xi, \theta_{\mathbf{C}}) \end{array}$$

Action contexts are defined by the following grammar, where $P \in \mathsf{K}$:

$$\mathbf{K} \quad ::= \quad \cdot \mid P \mid \overline{\nu}a.\mathbf{K} \mid a \bullet \mathbf{K} \mid \mathbf{K} \mid \mathbf{K}$$

We use A, B, C and their decorated variants to range over actions. We use the following conventions and definitions:

- For no ξ, θ, $\mathbf{C}$ do we have $\xi\theta_{\mathbf{C}} = \tau$, or $\xi\theta_{\mathbf{C}} = \epsilon$.
- If $\widetilde{a}$ is an empty vector of names, then $\overline{\nu}a.A = A$ and $\overline{\nu}a.\mathbf{K} = \mathbf{K}$. If $\widetilde{a} = a_1 \ldots a_n$, we identify $\widetilde{a}$ with the set $\{a_1, \ldots, a_n\}$.
- Let θ be a substitution mapping identifiers to Kell calculus terms. The support of θ, $\theta.\mathsf{supp}$, is $\{u \in \mathsf{L} \mid u\theta \neq u\}$. The co-support of θ, $\theta.\mathsf{cosupp}$, is $\{u\theta \mid u \in \theta.\mathsf{supp}\}$. The names of θ, $\theta.\mathsf{n}$, is $\{a \in \mathsf{N} \mid a \in \theta.\mathsf{supp} \cup \mathsf{fn}(\theta.\mathsf{cosupp})\}$. If θ is a context-dependent substitution and $\mathbf{C}$ is a Kell calculus context, note that $\theta_{\mathbf{C}}$ is also a substitution; therefore we can likewise define $\theta_{\mathbf{C}}.\mathsf{supp}$ and $\theta_{\mathbf{C}}.\mathsf{cosupp}$.
- We consider actions of the form $(\xi, \theta_{\mathbf{C}})$ only with the constraint that $\mathsf{bn}(\xi) \cup \mathsf{bv}(\xi) = \theta.\mathsf{supp}$, i.e. the substitution θ operates only on bound identifiers in ξ.

- By definition, ϵ is considered a neutral element for the parallel composition of actions, i.e. for all A, $A \mid \epsilon = \epsilon \mid A = A$. Also, we set $\overline{\nu}a.\epsilon = \epsilon$, $\overline{\nu}a.\tau = \tau$, $\overline{\nu}a.\langle A \rangle = \langle \overline{\nu}a.A \rangle$, $\langle \tau \rangle = \tau$, $\langle \epsilon \rangle = \epsilon$.
- For all A, $\lfloor A \rfloor$ is defined by induction as:

$$\begin{array}{lll} \lfloor \epsilon \rfloor = \epsilon & \lfloor \tau \rfloor = \tau & \lfloor \alpha \rfloor = \alpha \\ \lfloor \beta \rfloor = \beta & \lfloor \langle A \rangle \rfloor = A & \lfloor A \mid B \rfloor = \lfloor A \rfloor \mid \lfloor B \rfloor \\ \lfloor \overline{\nu}a.A \rfloor = \nu a.\lfloor A \rfloor & & \end{array}$$

- For all A, $|A|$ is defined by induction as:

$$\begin{array}{lll} |\epsilon| = 0 & |\tau| = 0 & |\alpha| = 0 \\ |\beta| = 0 & |\langle A \rangle| = |A| + 1 & |A \mid B| = \max(|A|, |B|) \\ |\overline{\nu}a.A| = |A| & & \end{array}$$

- For all A, $\lceil A \rceil$ is defined by induction as:

$$\begin{array}{lll} \lceil \epsilon \rceil = \epsilon & \lceil \tau \rceil = \tau & \lceil \alpha \rceil = \alpha \\ \lceil \beta \rceil = \beta & \lceil \langle A \rangle \rceil = \langle \lceil A \rceil \rangle & \lceil A \mid B \rceil = \lceil A \rceil \mid \lceil B \rceil) \\ \lceil \overline{\nu}a.A \rceil = \lceil A \rceil & & \end{array}$$

- For all A, the set of free names of A, $\mathtt{fn}(A)$, is defined by induction as:

$$\begin{array}{ll} \mathtt{fn}(\epsilon) = \emptyset & \mathtt{fn}(\tau) = \emptyset \\ \mathtt{fn}(a * P) = \mathtt{fn}(P) \cup \{a\} & \mathtt{fn}((\xi, \theta_{\mathbf{C}})) = \mathtt{fn}(\xi) \cup \mathtt{fn}(\theta_{\mathbf{C}}.\mathtt{cosupp}) \\ \mathtt{fn}(\langle A \rangle) = \mathtt{fn}(A) & \mathtt{fn}(A \mid B) = \mathtt{fn}(A) \cup \mathtt{fn}(B) \\ \mathtt{fn}(\overline{\nu}a.A) = \mathtt{fn}(A) \setminus \{a\} & \end{array}$$

- For all A, the set of free output names of A, $\mathtt{fon}(A)$, is defined by induction as:

$$\begin{array}{ll} \mathtt{fon}(\epsilon) = \emptyset & \mathtt{fon}(\tau) = \emptyset \\ \mathtt{fon}(a * P) = \mathtt{fn}(P) \cup \{a\} & \mathtt{fon}((\xi, \theta_{\mathbf{C}})) = \emptyset \\ \mathtt{fon}(\langle A \rangle) = \mathtt{fon}(A) & \mathtt{fon}(A \mid B) = \mathtt{fon}(A) \cup \mathtt{fon}(B) \\ \mathtt{fon}(\overline{\nu}a.A) = \mathtt{fon}(A) \setminus \{a\} & \end{array}$$

- For all A, the set of bound names of A, $\mathtt{bn}(A)$, is defined by induction as:

$$\begin{array}{ll} \mathtt{bn}(\epsilon) = \emptyset & \mathtt{bn}(\tau) = \emptyset \\ \mathtt{bn}(\alpha) = \emptyset & \mathtt{bn}(\beta) = \emptyset \\ \mathtt{bn}(\langle A \rangle) = \mathtt{bn}(A) & \mathtt{bn}(A \mid B) = \mathtt{bn}(A) \cup \mathtt{bn}(B) \\ \mathtt{bn}(\overline{\nu}a.A) = \mathtt{bn}(A) \cup \{a\} & \end{array}$$

- For all action contexts $\mathbf{K}$, $\lceil \mathbf{K} \rceil$ is defined by induction as:

$$\begin{array}{ll} \lceil \cdot \rceil = \cdot & \lceil P \rceil = P \\ \lceil a \bullet \mathbf{K} \rceil = a \bullet \lceil \mathbf{K} \rceil & \lceil \mathbf{K}_1 \mid \mathbf{K}_2 \rceil = \lceil \mathbf{K}_1 \rceil \mid \lceil \mathbf{K}_2 \rceil \\ \lceil \overline{\nu} a.\mathbf{K} \rceil = \lceil \mathbf{K} \rceil & \end{array}$$

- For all action contexts $\mathbf{K}$, the set of free names of $\mathbf{K}$, $\mathsf{fn}(\mathbf{K})$, is defined by induction as:

$$\begin{array}{ll} \mathsf{fn}(\cdot) = \emptyset & \mathsf{fn}(\mathbf{K}) = \mathsf{fn}(P) \quad \text{if } \mathbf{K} = P \\ \mathsf{fn}(a \bullet \mathbf{K}) = \mathsf{fn}(\mathbf{K}) \cup \{a\} & \mathsf{fn}(\mathbf{K} \mid \mathbf{K}') = \mathsf{fn}(\mathbf{K}) \cup \mathsf{fn}(\mathbf{K}') \\ \mathsf{fn}(\overline{\nu} a.\mathbf{K}) = \mathsf{fn}(\mathbf{K}) \setminus \{a\} & \end{array}$$

- We extend the structural congruence $\equiv$ on K to actions and action contexts in the following way:
  - If $A = B$, then $A \equiv B$.
  - If $A =_\alpha B$, then $A \equiv B$. If $\mathbf{K} =_\alpha \mathbf{K}'$, then $\mathbf{K} \equiv \mathbf{K}'$.
  - The parallel operator is associative and commutative.
  - If $\xi \equiv \xi'$ and $\mathbf{C}' \equiv \mathbf{E}[\mathbf{C}]$, then $(\xi, \theta_{\mathbf{C}}) \equiv (\xi', \theta_{\mathbf{C}'})$.
  - If $A \equiv B$, then $\langle A \rangle \equiv \langle B \rangle$. If $\mathbf{K} \equiv \mathbf{K}'$, then $a \bullet \mathbf{K} \equiv a \bullet \mathbf{K}'$
  - If $A \equiv B$, then $A \mid C \equiv B \mid C$. If $\mathbf{K} \equiv \mathbf{K}'$, then $\mathbf{K} \mid \mathbf{K}_0 \equiv \mathbf{K}' \mid \mathbf{K}_0$.
  - If $a \notin \mathsf{fn}(B)$, then $\overline{\nu} a.A \mid B \equiv (\overline{\nu} a.A) \mid B$. If $a \notin \mathsf{fn}(\mathbf{K}')$, then $\overline{\nu} a.\mathbf{K} \mid \mathbf{K}' \equiv (\overline{\nu} a.\mathbf{K}) \mid \mathbf{K}'$.
  - If $a \notin \mathsf{fon}(B)$ and if $\mathsf{bn}(\overline{\nu} a.A) \cap \mathsf{bn}(B)$, then $(\overline{\nu} a.A) \mid B \equiv \overline{\nu} a.A \mid B$.
  - If $A \equiv B$, then $\overline{\nu} a.A \equiv \overline{\nu} a.B$. If $\mathbf{K} \equiv \mathbf{K}'$, then $\overline{\nu} a.\mathbf{K} \equiv \overline{\nu} a.\mathbf{K}'$. For all A, $\mathbf{K}$, $\overline{\nu} a.A \equiv \nu a.A$ and $\overline{\nu} a.\mathbf{K} \equiv \nu a.\mathbf{K}$.
  - If $P \equiv Q$, then $\mathbf{K}[P] \equiv \mathbf{K}[Q]$.

With these conventions and definitions, note that if $A \neq \epsilon, \tau$, $|A| = 0$ and if no action β occurs in A, then $\lceil A \rceil$ is a Kell calculus process. Likewise, if $A \neq \epsilon, \tau$, $|A| \leq 1$ and if no action β occurs in A, then $\lceil [A] \rceil$ is a Kell calculus process. Also, if $\mathbf{K}$ is an action context, then $\lceil \mathbf{K} \rceil$ is a Kell calculus context.

The labelled transition system associated with Kell calculus processes is defined as the labelled transition system whose transition relation is the smallest relation satisfying the rules given in Figure 7, up to α-conversion (i.e. if $P =_\alpha P'$ and $Q' =_\alpha Q$ and $P' \xrightarrow{\mathbf{K} : A} Q'$, then $P \xrightarrow{\mathbf{K} : A} Q$).

We can now consider the correspondence between the reduction semantics and the labelled transition system semantics of the Kell calculus. We first have that the structural congruence $\equiv$ respects the transition relation. More precisely:

Theorem 1 *If $P \xrightarrow{\mathbf{K} : A} P'$ and $P \equiv Q$, then there exist Q', $\mathbf{K}'$, and B such that $Q \xrightarrow{\mathbf{K}' : B} Q'$, $P' \equiv Q'$, $\mathbf{K}' \equiv \mathbf{K}$, and $A \equiv B$.*

$$P \xrightarrow{P : \epsilon} P \ [\text{L.NULL}] \qquad a \bullet P \xrightarrow{a \bullet P : a \bullet P} \mathbf{0} \ [\text{L.ACT}] \qquad a \circ P \xrightarrow{a \circ P : a \circ P} \mathbf{0} \ [\text{L.PASS}]$$

$$\xi \triangleright P \xrightarrow{\cdot : (\xi, \theta_{\mathbf{C}})} P\theta_{\mathbf{C}} \ [\text{L.TRIG}] \qquad \frac{P \xrightarrow{\mathbf{K} : A} P' \qquad a \notin \mathsf{fn}(A)}{\nu a.P \xrightarrow{\overline{\nu} a.\mathbf{K} : A} \nu a.P'} \ [\text{L.NU.NF}]$$

$$\frac{P \xrightarrow{\mathbf{K} : A} P'}{a \bullet P \xrightarrow{a \bullet \mathbf{K} : \langle A \rangle} a \bullet P'} \ [\text{L.LOC}] \qquad \frac{P \xrightarrow{\mathbf{K} : A} P' \qquad a \in \mathsf{fon}(A)}{\nu a.P \xrightarrow{\overline{\nu} a.\mathbf{K} : \overline{\nu} a.A} P'} \ [\text{L.NU.F}]$$

$$\frac{\begin{array}{c} P_i \xrightarrow{\mathbf{K}_i : A_i} P_i' \ \ i = 0, 1 \qquad A_i = \tau \implies A_j = \epsilon \ \ i \neq j \\ \mathsf{bn}(A_1) \cap \mathsf{bn}(A_2) = \emptyset \qquad \cdot \text{ occurs at most once in } \mathbf{K}_1 \mid \mathbf{K}_2 \qquad \mathsf{bn}(A_i) \cap \mathsf{fn}(P_j) = \emptyset \ \ i \neq j \end{array}}{P_1 \mid P_2 \xrightarrow{\mathbf{K}_1 \mid \mathbf{K}_2 : A_1 \mid A_2} P_1' \mid P_2'} \ [\text{L.PAR}]$$

$$\frac{\begin{array}{c} P \xrightarrow{\mathbf{K} : A} P' \qquad \lceil A \rceil = A_1 \mid (\xi, \theta_{\mathbf{C}}) \mid A_2 \\ \xi : \mathsf{S} \qquad \mathbf{C} = \lceil \mathbf{K} \rceil \qquad |A_i| \leq 1 \ \ i = 1, 2 \qquad \xi\theta_{\mathbf{C}} \equiv [A_1 \mid A_2] \qquad \widetilde{c} = \mathsf{bn}(A) \end{array}}{P \xrightarrow{P : \tau} \nu \widetilde{c}.P'} \ [\text{L.RED.S}]$$

$$\frac{\begin{array}{c} P \xrightarrow{\mathbf{K} : A} P' \qquad \lceil A \rceil = A_3 \mid \langle A_1 \mid (\xi, \theta_{\mathbf{C}}) \mid A_2 \rangle \mid A_4 \qquad \xi : \mathsf{H} \qquad \mathbf{C} = \lceil \mathbf{K} \rceil \\ |A_i| \leq 1 \ \ i = 1, 2 \qquad |A_j| = 0 \ \ j = 3, 4 \qquad \xi\theta_{\mathbf{C}} \equiv A_3 \mid [A_1 \mid A_2] \mid A_4 \qquad \widetilde{c} = \mathsf{bn}(A) \end{array}}{P \xrightarrow{P : \tau} \nu \widetilde{c}.P'} \ [\text{L.RED.H}]$$

Figure 7: Transition rules

Proof: See Appendix. □

The exact correspondence between the reduction relation and the transition relation is given by the following theorem:

Theorem 2 *For all P, P', $P \xrightarrow{\mathbf{K} : \tau} \equiv P'$ if and only if $P \rightarrow P'$.*

Proof: See Appendix. □

3 Discussion

We begin this discussion by considering a simple pattern language for the Kell calculus. The syntax of simple patterns is given in Figure 8. Any name or process variable appearing in a simple pattern ξ is assumed to have a unique occurrence in ξ. Note that such patterns essentially correspond to

$$\begin{array}{lcl} \xi & ::= & \rho \mid \xi \mid \xi \\ \rho & ::= & a \mid a * u \mid a * \rho \mid a * (\rho \mid \rho) \\ u & ::= & a \mid (a) \mid (x) \end{array}$$

Figure 8: Syntax of simple patterns

$\mathsf{fn}(a) = \{a\}$	$\mathsf{fv}(a) = \emptyset$	$\mathsf{bn}(a) = \emptyset$	$\mathsf{bv}(a) = \emptyset$
$\mathsf{fn}(a * b) = \{a, b\}$	$\mathsf{fv}(a * b) = \emptyset$	$\mathsf{bn}(a * b) = \emptyset$	$\mathsf{bv}(a * b) = \emptyset$
$\mathsf{fn}(a * (b)) = \{a\}$	$\mathsf{fv}(a * (b)) = \emptyset$	$\mathsf{bn}(a * (b)) = \{b\}$	$\mathsf{bv}(a * (b)) = \emptyset$
$\mathsf{fn}(a * (x)) = \emptyset$	$\mathsf{fv}(a * (x)) = \emptyset$	$\mathsf{bn}(a * (x)) = \emptyset$	$\mathsf{bv}(a * (x)) = \{x\}$
$\mathsf{fn}(a * \xi) = \mathsf{fn}(\xi) \cup \{a\}$	$\mathsf{fv}(a * \xi) = \mathsf{fv}(\xi)$	$\mathsf{bn}(a * \xi) = \mathsf{bn}(\xi)$	$\mathsf{bv}(a * \xi) = \mathsf{bv}(\xi)$

Figure 9: Free names and bound markers for simple patterns

(parallel compositions of) polyadic messages. The resulting version of the Kell calculus is thus among the least expressive of the Kell calculus family. Nevertheless, as will be shown in this section, the higher-order kell operators yield considerable expressive power.

Free names (fn), free variables (fv), bound names (bn) and bound variables (bv) for simple patterns are defined in Figure 9. Matching for simple patterns is defined by extending standard substitutions θ on identifiers to context-dependent substitutions thus: $\forall \mathbf{C}\ \xi\theta_{\mathbf{C}} = \xi\theta$. The structural congruence relation on simple patterns is easily defined by the following rules: the parallel operator, $\mid$, is associative and commutative; if two patterns ξ and ζ are equivalent, $\xi \equiv \zeta$, then so are $a * \xi$ and $a * \zeta$; if two patterns differ only by α-conversion of their bound names or bound variables, then they are equivalent. Finally, all simple patterns are of sort S.

In this section, we consider, among other things: the definition of receptive triggers; the ability to define various forms of mobility and control, including Mobile Ambients-like constructs; encodings of the π-calculus and of the λ-calculus; the definition of concurrent objects.

3.1 Messages and reductions

As discussed in the introduction, kells in the Kell calculus can serve both as locations and as messages. For example, the configuration C below has two reductions:

$$C = a \bullet (b \bullet x \triangleright \mathbf{0}) \mid b \bullet (a \bullet x \triangleright \mathbf{0})$$

$$C \to a \bullet \mathbf{0} \qquad C \to b \bullet \mathbf{0}$$

In the first reduction, kell a is the locus of computation, and kell b plays the role of a message, received by the trigger in a. In the second reduction, the roles of a and b are reversed.

Note that processes reduced to a simple name can also be received by triggers. Thus, we have the following reduction: $a \mid (a \triangleright a) \to a$. (Alternatively, one can think of the process a as being an abbreviation for the process $a \bullet \mathbf{0}$).

Simple patterns do not distinguish between the position of messages in an execution context. For instance, if pattern ξ matches message M, i.e. there is a substitution θ such that $\xi\theta = M$, then we have the following reductions:

$$\begin{array}{rcl} a \bullet (\xi \triangleright P \mid Q_h) \mid M & \rightarrow & a \bullet (P\theta \mid Q_h) \\ \xi \triangleright P \mid M \mid Q & \rightarrow & P\theta \mid Q \\ \xi \triangleright P \mid a \bullet (M \mid Q_l) & \rightarrow & P\theta \mid a \bullet Q_l \end{array}$$

It is however possible to obtain the effect of a pattern language that distinguishes between the above three situations by using kells which act as wrappers. If ξ is a simple pattern, define $\overline{\xi}$ by:

$$\begin{array}{rr} \overline{a} = a & \overline{a * a} = a * a \\ \overline{a * (b)} = a * b & \overline{a * x} = a * x \\ \overline{a * \xi} = a * \overline{\xi} & \overline{\xi_1 \mid \xi_2} = \overline{\xi_1} \mid \overline{\xi_2} \end{array}$$

We can obtain a distinction between the different situations above with the following high and low wrappers:

$$\nu h\, l.\, h \bullet (\mathtt{HR} \mid Q_h \mid a \bullet (\mathtt{high}\langle \xi \rangle \triangleright P \mid \mathtt{low}\langle \xi \rangle \triangleright P \mid \xi \triangleright P \mid Q \mid l \bullet (\mathtt{LR} \mid b \bullet Q_l)))$$

where HR and LR are simple routers defi ned by:

$$\mathtt{HR} = \xi \triangleright \mathtt{high}\langle \overline{\xi} \rangle \qquad \mathtt{LR} = \xi \triangleright \mathtt{low}\langle \overline{\xi} \rangle$$

The two routers simply transfer messages on special channels high and low, which help indicate to the actual receiver the origin of messages (from the outside of the kell or from a subkell).

3.2 Receptive triggers

As can be seen in rule R.RED.S, a trigger disappears after it has been triggered. Because the calculus is higher-order, however, it is possible to define *receptive triggers*, i.e. triggers that are preserved during a reduction (similar to Join calculus definitions). Receptive triggers are important because they provide a way to define recursive processes (receptive triggers correspond to Join calculus definitions and to replicated output in the π-calculus). It is known from the work on CHOCS [27] that recursion can be defined, in a higher-order setting, by means of process passing and communication. We show below how to define receptive triggers in the Kell calculus. Let t, ξ and P be such that $t \notin \mathtt{fn}(\xi) \cup \mathtt{fn}(P)$, and define $Y(P, \xi, t)$ by:

$$Y(P, \xi, t) = \xi \mid t \circ y \triangleright P \mid y \mid t \circ y$$

Let $A(P, \xi) = \nu t.\, Y(P, \xi, t) \mid t \circ Y(P, \xi, t)$. Let $\mathbf{C}$, Q, Q', R, M, N and θ be as in the premises of rule R.RED.H. By rule S.α, it is always possible to choose t such that $t \notin \mathtt{fn}(N) \cup \mathtt{fn}(M) \cup \mathtt{fn}(Q) \cup \mathtt{fn}(R) \cup \mathtt{fn}(P\theta_{\mathbf{C}})$. Then, by rules S.NU.KELL and S.NU.PAR, we have:

$$a \bullet (A(P, \xi) \mid N \mid Q \mid R) \mid M \equiv \nu t.\, a \bullet (Y(P, \xi, t) \mid t \circ Y(P, \xi, t)) \mid N \mid Q \mid R) \mid M$$

By construction and by rule R.RED.H we have: $\nu t.\, a \bullet (Y(P, \xi, t) \mid t \circ Y(P, \xi, t)) \mid N \mid Q \mid R) \mid M \rightarrow \nu t.\, a \bullet (P\theta_{\mathbf{C}} \mid Y(P, \xi, t) \mid t \circ Y(P, \xi, t)) \mid Q' \mid R)$. Since $t \notin \mathtt{fn}(P) \cup \mathtt{fn}(Q) \cup \mathtt{fn}(P\theta_{\mathbf{C}})$, we

$$\frac{\mathbf{K} = \cdot \mid N \mid Q \qquad Q = \prod_{j\in J} a_j \bullet (N_j \mid Q_j) \qquad Q' = \prod_{j\in J} a_j \bullet Q_j \qquad \xi\theta_{\mathbf{C}} \equiv N \mid \prod_{j\in J} N_j}{\xi : \mathsf{S} \qquad \xi \diamond P \mid N \mid Q \to \xi \diamond P \mid P\theta_{\mathbf{C}} \mid Q'} \text{ [R.REC.S]}$$

$$\frac{Q = \prod_{j\in J} a_j \bullet (N_j \mid Q_j) \qquad Q' = \prod_{j\in J} a_j \bullet Q_j \qquad \xi\theta_{\mathbf{C}} \equiv M \mid N \mid \prod_{j\in J} N_j}{\xi : \mathsf{H} \qquad \mathbf{K} = M \mid a \bullet (\cdot \mid Q \mid R) \qquad a \bullet (\xi \diamond P \mid N \mid Q \mid R) \mid M \to a \bullet (\xi \diamond P \mid P\theta_{\mathbf{C}} \mid Q' \mid R)} \text{ [R.REC.H]}$$

Figure 10: Derived reduction rules II

have $\nu t.\, a \bullet (P\theta_{\mathbf{C}} \mid Y(P,\xi,t) \mid t \circ Y(P,\xi,t) \mid Q' \mid R) \equiv a \bullet (P\theta_{\mathbf{C}} \mid \nu t.Y(P,\xi,t) \mid t \circ Y(P,\xi,t) \mid Q' \mid R) = a \bullet (P\theta_{\mathbf{C}} \mid A(P,\xi) \mid Q' \mid R)$ by rules S.NU.KELL and S.NU.PAR. Thus, by rules R.CONTEXT and R.EQUIV,

$$a \bullet (A(P,\xi) \mid N \mid Q \mid R) \mid M \to a \bullet (A(P,\xi) \mid P\theta_{\mathbf{C}} \mid Q' \mid R)$$

We can reason similarly with rule R.RED.S. We have just shown that the construct $A(P,\xi)$ acts exactly as a receptive trigger. More precisely, let us note $\xi \diamond P$ the construct $A(P,\xi)$: we have just shown that the reduction rules R.REC.S and R.REC.H in Figure 10 are derived rules of the Kell calculus.

3.3 Encoding the π-calculus and the λ-calculus

The π-calculus and the λ-calculus constitute standards of expressive power. They can be simply and directly encoded in the Kell calculus. This is obvious in the case of the asynchronous π-calculus, but, because of its higher-order character, the Kell calculus can also encode directly the synchronous π-calculus. The synchronous (polyadic) π-calculus with positive name matching and input guarded sums (cf [24] for a definition) is indeed a direct sub-calculus of the Kell calculus. A simple encoding is given below, where we assume that the names $1, \ldots, n, \ldots$, and ok do not appear free in P, P_j, Q, and where $\widetilde{b} = b_1 \ldots b_n$, $\widetilde{b^j} = b_1^j \ldots b_{n_j}^j$, $j \in J$.

$$\begin{aligned}
[\![a\widetilde{b}.P]\!] &= a\langle [\![P]\!], b_1, \ldots, b_n\rangle \\
[\![\tau.P]\!] &= \nu \mathsf{k}.\mathsf{k} \mid (\mathsf{k} \triangleright [\![P]\!]) \\
[\![[a=b]P]\!] &= \nu l.(l \circ a \triangleright [\![P]\!]) \mid l \circ b \\
[\![\nu a.P]\!] &= \nu a.[\![P]\!] \\
[\![P \mid Q]\!] &= [\![P]\!] \mid [\![Q]\!] \\
[\![!P]\!] &= \nu \mathsf{k}.\mathsf{k} \mid (\mathsf{k} \diamond [\![P]\!] \mid \mathsf{k}) \\
[\![\sum_{j\in J} a_j(\widetilde{b^j}).P_j]\!] &= \nu \mathsf{k}.\mathsf{k} \mid \prod_{j\in J} \mathsf{k} \mid a_j\langle x_j, (b_1^j), \ldots, (b_{n_j}^j)\rangle \triangleright [\![P_j]\!] \mid x_j
\end{aligned}$$

An encoding of the λ-calculus in the Kell calculus can be obtained indirectly through the encoding of the π-calculus, however a simpler and more direct one is given below, where we assume that the names λ, $\mathtt{fun}$, $\mathtt{arg}$, and $\mathtt{abs}$, do not appear free in P, Q:

$$\begin{array}{rcl} [\![x]\!] & = & x \\ [\![\lambda x.P]\!] & = & \mathtt{abs} \circ (\lambda \circ x \triangleright [\![P]\!]) \\ [\![PQ]\!] & = & \nu \mathtt{fun}\, \mathtt{arg}.\, \mathtt{App} \mid \mathtt{fun} \bullet [\![P]\!] \mid \mathtt{arg} \circ [\![Q]\!] \\ \mathtt{App} & = & \mathtt{fun} \bullet (\mathtt{abs} \circ p) \mid \mathtt{arg} \circ q \triangleright p \mid \lambda \circ q \end{array}$$

We have the following reductions:

$$\begin{array}{rcl} [\![(\lambda x.P)Q]\!] & = & \nu \mathtt{fun}\, \mathtt{arg}.\, \mathtt{App} \mid \mathtt{fun} \bullet (\mathtt{abs} \circ (\lambda \circ x \triangleright [\![P]\!])) \mid \mathtt{arg} \circ [\![Q]\!] \\ & \rightarrow & \nu \mathtt{fun}\, \mathtt{arg}.\lambda \circ x \triangleright [\![P]\!] \mid \lambda \circ [\![Q]\!] \\ & \rightarrow & \nu \mathtt{fun}\, \mathtt{arg}.[\![P]\!]\{[\![Q]\!]/x\} \\ & \equiv & [\![P]\!]\{[\![Q]\!]/x\} \end{array}$$

It is easy to prove by induction on the structure of P that $[\![P\{Q/x\}]\!] \equiv [\![P]\!]\{[\![Q]\!]/x\}$, hence we have: $[\![(\lambda x.P)Q]\!] \rightarrow^2 \equiv [\![P\{Q/x\}]\!]$. Note that the above encoding enforces as call-by-name evaluation strategy.

3.4 Defining objects

With the kell construct as a generalized (named) record construct, one can obtain various forms of objects directly. To illustrate, we show here how one can simulate concurrent objects directly inspired by the concurrent object calculus of Gordon and Hankin [14], itself inspired by the object calculi introduced by Abadi and Cardelli in [1]. This simulation is quite close to the encoding of the Gordon and Hankin concurrent object calculus in the Blue calculus with records proposed in [31].

We consider objects to consist of named records of the following form ($I_n = \{1, \ldots, n\}$):

$$a \mapsto [l_i = \zeta(x_i) f_i^{i \in I_n}]$$

where a is the reference of the object, and l_i are the methods of the object. There are three basic operations which are available on objects: invoking a method $a.l_j$, updating a method $a.l_j \leftarrow \zeta(x)f$, and cloning an object $\mathtt{clone}(a)$. The informal operational semantics of these three constructs is given by the following rules, inspired by the operational rules in [14] (for simplicity, we omit the continuations returned as part of the different operations). Let $d = [l_i = \zeta(x_i) f_i^{i \in I_n}]$ and $d' = [l_j = \zeta(x)f, l_i = \zeta(x_i) f_i^{i \in I_n \setminus \{j\}}]$, we have:

$$(a \mapsto d) \mid a.l_j \rightarrow (a \mapsto d) \mid f_j\{a/x_j\} \text{ [INVOKE]}$$

$$(a \mapsto d) \mid (a.l_j \leftarrow \zeta(x)f) \rightarrow (a \mapsto d) \mid (a \mapsto d') \text{ [UPDATE]}$$

$$(a \mapsto d) \mid \mathtt{clone}(a) \rightarrow (a \mapsto d) \mid \nu b.(b \mapsto d) \text{ [CLONE]}$$

Such behavior can be faithfully mimicked in the Kell calculus with simple patterns, using the following definitions, where we recall that $\mathtt{m}\langle V_1, \ldots, V_n \rangle$ is an abbreviation for $\mathtt{m} \circ (1 \circ V_1 \mid \ldots \mid n \circ V_n)$,

and where we assume that the names $1, \ldots, n, \ldots$, $\mathtt{m}$, $\mathtt{inv}$, $\mathtt{upd}$, $\mathtt{clone}$, $\mathtt{r}$ and $\mathtt{make}$ do not occur free in any of the f, f_i, l, l_i:

$$\begin{aligned}
[\![a \mapsto d]\!] &= \mathtt{Env} \mid \mathtt{Object}(a, l_i, \mathtt{Meth}(a, l_i, b_i, f_i))^{i \in I_n} \\
[\![a.l]\!] &= \mathtt{m}\langle a, \mathtt{inv}, l\rangle \\
[\![a.l \leftarrow \zeta(x)f]\!] &= \mathtt{m}\langle a, \mathtt{upd}, l, (\mathtt{r} \circ (l \circ x) \triangleright [\![f]\!] \mid\ a)\rangle \\
[\![\mathtt{clone}(a)]\!] &= \mathtt{m}\langle a, \mathtt{clone}\rangle
\end{aligned}$$

with the definitions:

$$\begin{aligned}
\mathtt{Object}(a, l_i, q_i)^{i \in I_n} &= a \bullet (\mathtt{Clone}(a) \mid a \mid \prod_{i \in I_n} \mathtt{Invoke}_i(a) \mid \mathtt{Update}_i(a) \mid l_i \circ q_i \\
\mathtt{Invoke}_i(a) &= \mathtt{m}\langle a, \mathtt{inv}, l_i\rangle \mid l_i \circ q) \mid\ a \diamond \mathtt{r}\langle l_i \circ a\rangle \mid q \mid l_i \mid l_i \circ q \\
\mathtt{Update}_i(a) &= \mathtt{m}\langle a, \mathtt{upd}, l_i, p\rangle \mid l_i \circ q \mid\ a \diamond l_i \circ p \mid\ a \\
\mathtt{Clone}(a) &= \mathtt{m}\langle a, \mathtt{clone}\rangle \mid\ a \mid \prod_{i \in I_n} l_i \circ q_i \diamond \\
&\qquad \prod_{i \in I_n} l_i \circ q_i \mid\ a \mid\ \nu b.\, \mathtt{make}\langle b, l_i, q_i\rangle^{i \in I_n} \\
\mathtt{Meth}(a, l, b, f) &= \mathtt{r}\langle l \circ (b)\rangle \triangleright [\![f]\!] \mid\ a
\end{aligned}$$

and where the environment process $\mathtt{Env}$ for routing messages between objects and activating cloned objects, is defined thus:

$$\begin{aligned}
\mathtt{Env} &= \mathtt{Router} \mid \mathtt{Factory} \\
\mathtt{Router} &= (\mathtt{m} \circ x \diamond \mathtt{m} \circ x) \\
\mathtt{Factory} &= \mathtt{make}\langle (b), (l_i), q_i\rangle^{i \in I_n} \diamond \mathtt{Object}(b, l_i, q_i)^{i \in I_n}
\end{aligned}$$

The process $\mathtt{Router}$ just helps forward messages between objects. The process $\mathtt{Factory}$ is responsible for creating new objects upon request.

3.5 Process control

Notice that the context $(a \bullet \cdot)$ is an execution context. Processes in this context can freely execute. As rule IN reveals, they can also receive messages from their environment. Combining trigger and kell constructs provides different degrees of control over the execution of a process:

- Within $a \circ P$, process P is passivated, but it remains available for reactivation by its environment. Reactivation can take place, for instance, in a context of the form $\cdot \mid (a \circ x \triangleright x)$, where the following reduction is valid:

$$a \circ P \mid (a \circ x \triangleright x) \to P$$

- Within $a \bullet P$, process P is active, and can communicate with its environment. In a context of the form $a \bullet x \triangleright a \circ x \mid\ \cdot$, process P can be passivated through the following reduction:

$$(a \bullet x \triangleright a \circ x) \mid a \bullet P \to a \circ P$$

$a \bullet P$ is also a message that allows process P to move from kell to kell while remaining active. Thus, $a \bullet P$ can model a mobile agent that can interact with its environment (if P contains at least one trigger) while roaming a network of kells.

- Within $b \bullet (a \bullet P)$, process P is active but it cannot communicate with its environment. This is another example of a mobile agent, however in this case the agent is prevented from interacting with its environment while moving.
- With simple patterns, within $\nu a\, b.b \bullet (a \bullet P)$, with $a, b \notin \mathtt{fn}P$, process P is active but it is totally isolated from its environment: it can no longer communicate with it and can no longer be manipulated by it.
- With simple patterns, within $\nu a\, b.b \bullet (a \circ P)$, process P is inactive, and can no longer be manipulated by its environment.

Note that, with simple patterns, there are several forms of processes, apart from $\nu a\, b.b \bullet (a \circ P)$, that can be intuitively equated with $\mathbf{0}$. This is the case, for instance, of $\nu a\, b.b \bullet (\xi \mid a \bullet x \triangleright P)$. Even though pattern ξ may match external messages, the whole pattern expects a message of the form $a \bullet Q$ which can never be present. Thus, there is no Kell calculus context with simple patterns that can distinguish between $\nu a\, b.b \bullet (\xi \mid a \bullet x \triangleright P)$ and $\mathbf{0}$.

Each kell in the Kell calculus can be endowed with its own control behavior (with respect to the processes it contains) through the use of the box operators. Consider a kell of the form $a \bullet (P \mid \mathtt{c} \bullet Q)$. Through the $\mathtt{c}$ handle, process P can control the execution of Q. For instance, if $P = \mathtt{kill}\langle a \rangle \mid \mathtt{c} \bullet q \triangleright \mathbf{0}$, then we have:

$$\mathtt{kill}\langle a \rangle \mid a \bullet (P \mid \mathtt{c} \bullet Q) \rightarrow a \bullet \mathbf{0}$$

Message $\mathtt{kill} \bullet a$ can be seen as an explicit termination command or as modelling the occurrence of a failure of kell a. Along the same lines, with

$$P = (\mathtt{suspend}\langle a \rangle \mid \mathtt{c} \bullet q) \diamond \mathtt{c} \circ q \qquad R = (\mathtt{resume}\langle a \rangle \mid \mathtt{c} \circ q) \diamond \mathtt{c} \bullet q$$

we have the following reductions:

$$\mathtt{suspend}\langle a \rangle \mid a \bullet (P \mid R \mid \mathtt{c} \bullet Q) \rightarrow a \bullet (P \mid R \mid \mathtt{c} \circ Q)$$

$$\mathtt{resume}\langle a \rangle \mid a \bullet (P \mid R \mid \mathtt{c} \circ Q) \rightarrow a \bullet (P \mid R \mid \mathtt{c} \bullet Q)$$

which mean that $(P \mid R)$ can suspend or resume the execution of Q, depending on whether Q is active or suspended.

Notice that it is impossible for a kell $a \bullet P$ to "commit suicide", i.e. to become $\mathbf{0}$ by means of purely internal reductions of process P, as the kill example above suggests. Indeed, if $P \rightarrow Q$ then we have $a \bullet P \rightarrow a \bullet Q$, so the best that can be achieved is a reduction to $a \bullet \mathbf{0}$, which, as a message, can still be handled by the environment. However, a kell can signal its willingness to be garbage collected, e.g. by emitting a message of the form $\mathtt{collect}\langle \ldots \rangle$. A process such as process `Collector` below can be used for that purpose. Note that it relies on the cooperation of the kell to

be collected and on an authentication scheme using a password k (there could be several kells named a in a given context).

$$\begin{array}{rcl}
\texttt{Collector} & = & \texttt{collect}\langle(a),(k),p\rangle \diamond (a \bullet x \triangleright \\
 & & \nu r\, b.\, \texttt{F}(a,k,r) \mid b \bullet (a \bullet x \mid \texttt{query}\langle a,k,r\rangle \mid r\langle a,k,(b)\rangle \triangleright r\langle a,k,b\rangle)) \\
\texttt{F}(a,k,r) & = & (r\langle a,k,\texttt{yes}\rangle \triangleright (b \bullet y \triangleright p)) \\
 & & \mid (r\langle a,k,\texttt{no}\rangle \triangleright (b \bullet y \triangleright \texttt{collect}\langle a,k\rangle \mid a \bullet x))
\end{array}$$

3.6 Encoding Mobile Ambients

The combination of box operators and higher-order communication provides for various forms of mobility. For instance, the Kell calculus can emulate the different mobility primitives of Mobile Ambients [7]. The encoding given below illustrates this. The encoding is deadlock-free, but it relies on a simple locking scheme that reduces the parallelism inherent in ambient reductions, and it exhibits divergence (potentially infinite idle reduction loops). An encoding that does not suffer from these latter two limitations is certainly possible (e.g. one could mimick the protocol employed in the Join calculus implementation of ambients described in [12]) but it would be more complex. The encoding uses the process `Collector` defined above to garbage collect kells that have been opened, and λ-calculus constructs (λ abstraction and application) as defined in Section 3.3. Since, in Mobile Ambients, there can be several ambients bearing the same name in parallel, authentication by `Collector` is required prior to collecting a given ambient. In order to simplify the presentation of the encoding, we will present it using an extension of simple patterns, namely simple patterns with direction. This pattern language allows trigger patterns to discriminate between messages coming from the outside of the receiving kell, and from messages originating from a subkell of the receiving kell. We have seen in Section 3.1 how such a discrimination could be realized with simple patterns. Using similar constructs, we can thus obtain an encoding of Mobile Ambients into the Kell calculus with simple patterns. However, the encoding below is more readable. Simple patterns with direction are defined by the following grammar:

$$\xi \quad ::= \quad \rho \quad \mid \quad \rho^{\uparrow} \quad \mid \quad \rho_{\downarrow} \quad \mid \quad \xi \,|\, \xi$$

where ρ refers to the ρ productions in the grammar of simple patterns in Figure 8. Simple patterns with direction are a subset of advanced patterns defined, together with the associated semantics of matching, in Section 4.1.

The encoding of Mobile Ambients in the Kell calculus with simple patterns (with direction) is defined as follows.

$$\begin{array}{ll}
[\![\mathbf{0}]\!] = \mathbf{0} & [\![\texttt{in}\, a.P]\!] = \texttt{in}\langle a, [\![P]\!]\rangle \\
[\![\nu n.P]\!] = \nu n.[\![P]\!] & [\![\texttt{out}\, a.P]\!] = \texttt{out}\langle a, [\![P]\!]\rangle \\
[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!] & [\![\texttt{open}\, a.P]\!] = \texttt{open}\langle a, [\![P]\!]\rangle \\
[\![!P]\!] = \nu a.\, a \mid (a \triangleright [\![P]\!] \mid a) & \\
[\![a[P]]\!] = a \bullet (\texttt{A}(a) \mid \texttt{amb} \bullet [\![P]\!]) \mid \texttt{AmbEnv} &
\end{array}$$

$$
\begin{array}{rcl}
\mathsf{A}(a) & = & \nu \mathsf{t}.\, \mathsf{t} \mid \mathsf{S}(a,\mathsf{t}) \mid \mathsf{T}(a,\mathsf{t}) \mid \mathsf{F}(a,\mathsf{t}) \mid \mathsf{NQ}(a,t) \\
\mathsf{S}(a,\mathsf{t}) & = & (\mathsf{t} \mid \mathsf{in}\langle b,p\rangle_{\downarrow} \triangleright \\
 & & \quad \mathsf{amb} \bullet z \triangleright \nu k.\, \mathsf{collect}\langle a,k,\mathsf{to}\langle a,\mathsf{in},m,p,z\rangle\rangle \mid \mathsf{YQ}(a,k)) \\
 & & \mid (\mathsf{t} \mid \mathsf{out}\langle b,p\rangle_{\downarrow} \triangleright \\
 & & \quad \mathsf{amb} \bullet z \triangleright \nu k.\, \mathsf{collect}\langle a,k,\mathsf{up} \circ \langle a,\mathsf{out},m,p,z\rangle\rangle \mid \mathsf{YQ}(a,k)) \\
\mathsf{T}(a,\mathsf{t}) & = & (\mathsf{t} \mid \mathsf{amb} \bullet z \mid \mathsf{open}\langle a,p\rangle^{\uparrow} \triangleright \\
 & & \quad \nu k.\, \mathsf{collect}\langle a,k,(z \mid p)\rangle \mid \mathsf{YQ}(a,k)) \\
 & & \mid (\mathsf{t} \mid \mathsf{to}\langle n,\mathsf{in},a,p,x\rangle^{\uparrow} \mid \mathsf{amb} \bullet z \diamond \\
 & & \quad (\nu k.\, \mathsf{make}\langle n,p \mid x,k\rangle \mid (k\langle y\rangle^{\uparrow} \triangleright t \mid \mathsf{amb} \bullet (z \mid y))) \\
 & & \mid (\mathsf{t} \mid \mathsf{up}\langle n,\mathsf{out},a,p,x\rangle_{\downarrow} \diamond \\
 & & \quad \mathsf{amb} \bullet z \triangleright (\nu k.\, \mathsf{make}\langle n,p \mid x,k\rangle \mid (k\langle y\rangle^{\uparrow} \triangleright t \mid \mathsf{amb} \bullet z))) \\
\mathsf{YQ}(a,k) & = & \mathsf{query}\langle a,k,(r)\rangle \triangleright r\langle a,k,\mathsf{yes}\rangle \\
\mathsf{NQ}(a,t) & = & \mathsf{t} \mid \mathsf{query}\langle a,(k),(r)\rangle \diamond \mathsf{t} \mid r\langle a,k,\mathsf{no}\rangle \\
\mathsf{F}(a,\mathsf{t}) & = & (\mathsf{t} \mid \mathsf{to}\langle n,\mathsf{in},m,p,x\rangle_{\downarrow} \diamond \\
 & & \quad \mathsf{amb} \bullet z \triangleright \\
 & & \quad (\nu k.\, \mathsf{make}\langle n,\mathsf{in}\langle m,p\rangle \mid x,k\rangle \mid (k\langle y\rangle^{\uparrow} \triangleright t \mid \mathsf{amb} \bullet (z \mid y))) \\
 & & \mid (\mathsf{t} \mid \mathsf{up}\langle n,\mathsf{out},m,p,x\rangle_{\downarrow} \diamond \\
 & & \quad \mathsf{amb} \bullet z \triangleright \\
 & & \quad (\nu k.\, \mathsf{make}\langle n,\mathsf{out}\langle m,p\rangle \mid x,k\rangle \mid (k\langle y\rangle^{\uparrow} \triangleright t \mid \mathsf{amb} \bullet (z \mid y))) \\
\mathsf{AmbEnv} & = & \nu \mathsf{t}.\, (\mathsf{C}\ \mathsf{t}\ \mathsf{C}) \mid \mathsf{Collector} \\
\mathsf{C} & = & \lambda \mathsf{t}\, f.\, \mathsf{Factory}(\mathsf{t}) \mid \mathsf{t}\langle f\rangle \\
\mathsf{Factory}(\mathsf{t}) & = & \mathsf{t}\langle f\rangle \mid \mathsf{make}\langle n,p,k\rangle_{\downarrow} \diamond \\
 & & \quad k\langle \nu a.(f\ a\ f) \mid \mathsf{Collector} \mid n \bullet (\mathsf{A}(n) \mid \mathsf{amb} \bullet p)\rangle
\end{array}
$$

A few comments on this encoding are in order. The encoding of the ambient construct, $a[P]$, is typical of encoding of calculi with explicit locations. The process $\mathsf{A}(a)$ in the encoding can be understood as implementing the interaction protocol that is characteristics of Mobile Ambients. Encoding of other forms of ambient calculi would involve defining different variants of this process. Process AmbEnv is a helper process that characterizes the environment required by Mobile Ambients, and that provides garbage collection and factory facilities. The auxiliary processes $\mathsf{A}(a)$ and AmbEnv are defined below. We use a lock t per ambient to avoid conflicts between concurrent moves. Process $\mathsf{S}(a,\mathsf{t})$ starts the execution of in and out moves at the source ambient. Process $\mathsf{T}(a,\mathsf{t})$ implements the open primitive and terminates the execution of in and out moves originated at a source ambient within process S. Notice that all the ambient primitives lead to the the destruction of the source ambient, which is later recreated at the end of execution of the in and out primitives. Process $\mathsf{F}(a,\mathsf{t})$ non-deterministically aborts transactions implementing the in and out moves to avoid unduly blocking an ambient (and thus preserving the mobile ambient semantics).

This process is the source of divergence in the encoding. Such divergence can be removed at the expense of a slightly more complex encoding, e.g. where each controller process $\mathsf{A}(a)$ maintains a list of names of current sub-ambients.

3.7 Distributed interpretation

At this point, some comments are in order to justify our belief that the Kell calculus constitutes a *distributed* process calculus. The main issue has to do with the implementability of the reduction rule R.RED.H in a distributed setting and the existence of potential *conflict* situations. Rule R.RED.H is local in that it involves a single receiver located at a single kell, but consider the following instance of derived rule IN:

$$a \bullet (b \bullet x \triangleright x \mid Q) \mid b \bullet P \rightarrow a \bullet (P \mid Q)$$

In this reduction, one must first decide that the receiving kell is the a kell and that b plays the role of a message (process P may contain triggers ready to receive messages of its own). However, note that the decision to consider b as a message is a purely local one. It involves deciding at b that b should be sent to the trigger at a and not considered for the reception of messages of its own. This looks like a standard execution or scheduling decision, which does not require a distributed consensus to take place, and which requires only the knowledge of the fact that there is a receiver ready for b located at a different kell at this level of the system. Such knowledge is in turn not dissimilar to the one that a given reference in a distributed system is in fact a reference to a server, which can receive messages. More problematic is the following situation, where it is possible to apply both derived rule BETA (with the trigger on a) and derived rule OUT (with the trigger on b):

$$a \bullet x \triangleright x \;\mid\; a \bullet (b\langle P \rangle \mid Q) \;\mid\; b \circ y \triangleright y$$

Here again, however, notice that to choose between the BETA reduction and the OUT reduction, all that is involved is a local choice at kell a, namely to send kell a to the trigger on a, or to send message b (which is clearly a message, in this case, since it corresponds to a passive kell) to the trigger on b. Provided a and b are identified as potential receivers, then only local actions are required to implement the possible reductions (a scheduling choice at kell a, and a reception at trigger a or at trigger b). A yet more problematic case occurs in the following situation:

$$c \bullet (a \bullet x \triangleright x) \;\mid\; a \bullet (b \bullet y \triangleright y \mid Q) \;\mid\; b \bullet P$$

In this case, there are two possible reductions, both relying on derived rule IN (the b kell received at kell a on the b trigger, and the a kell received at kell c on the a trigger). There is still the need of a local choice to make at kell a, either to send it to trigger a, or to make it available for the receipt of messages on trigger b. But the situation here is similar to the situation with the IN rule of Mobile Ambients, or, for that matter, with the `go` primitive of the Distributed Join calculus: moving kell b to kell a would require locking kell a in place in order to make the move atomic. This is made more evident in the following situation:

$$c \bullet (a \bullet x \triangleright x) \;\mid\; a \bullet (c \bullet x \triangleright x)$$

If both scheduling decisions at kell c and kell a are to consider the kells as messages, then the configuration is deadlocked whereas one would expect at least one of the two possible reductions

(using rule IN) to take place. We thus see the need to implement kell mobility in the general kell calculus by means of a distributed transaction mechanism with deadlock detection or prevention.

Fortunately, the potential cost of general kell mobility need not be paid for every communication as in Mobile Ambients, and we can have a truly distributed interpretation of the calculus. First, notice that passive kells $a \circ P$ should always be interpreted as standard asynchronous messages. No such conflict situations arise when communication between kells relies on passive kells. Thus, systems that do not require communication of active kells do not pay the cost of a distributed transaction for each communication. Obviously, routing an asynchronous message in presence of kell mobility is a non-trivial business, but in a model as general as the Kell calculus (or in an actual distributed system, for that matter), that is only to be expected. We can formalize this restriction of using passive kells only as messages crossing kell boundaries as a subcalculus of the Kell calculus, which we call dK. Messages that cross kell boundaries in dK take the form given by the following grammar:

$$\overline{M}, \overline{N} ::= \mathbf{0} \quad | \quad a \quad | \quad a \circ P \quad | \quad (\overline{M} \mid \overline{N})$$

The operational semantics of dK is given by the same structural congruence and reduction rules as the full Kell calculus, except that rules R.RED.S and R.RED.H are modified thus:

$$\frac{\xi : \mathsf{S} \qquad \mathbf{K} = \cdot \mid N \mid Q \qquad Q = \prod_{j \in J} a_j \bullet (\overline{N_j} \mid Q_j) \qquad Q' = \prod_{j \in J} a_j \bullet Q_j \qquad \xi\theta_{\mathbf{K}} = N \mid \prod_{j \in J} \overline{N_j}}{\xi \triangleright P \mid N \mid Q \to P\theta_{\mathbf{K}} \mid Q'} \text{ [DK.RED.S]}$$

$$\frac{\xi : \mathsf{H} \qquad \mathbf{K} = \overline{M} \mid a \bullet (\cdot \mid Q \mid R) \qquad Q = \prod_{j \in J} a_j \bullet (\overline{N_j} \mid Q_j) \qquad Q' = \prod_{j \in J} a_j \bullet Q_j \qquad \xi\theta_{\mathbf{K}} = \overline{M} \mid N \mid \prod_{j \in J} \overline{N_j}}{a \bullet (\xi \triangleright P \mid N \mid Q \mid R) \mid \overline{M} \to a \bullet (P\theta_{\mathbf{K}} \mid Q' \mid R)} \text{ [DK.RED.H]}$$

In dK, each active kell should be understood primarily as a locus of computation (a location), possibly under the control of a surrounding behavior (i.e. a set of triggers located immediately outside of the active kell under consideration). In this calculus, we can no longer have conflict situations such as the two last ones identified above, which do not have any reduction in dK. Certain conflicts still exist in dK, but they are harmless. As a first instance, consider configuration C below:

$$C = a \bullet x \triangleright \mathbf{0} \mid a \bullet (b \circ x \triangleright x) \mid b \circ P$$

Configuration C has the following two reductions:

$$C \to b \circ P \qquad C \to a \bullet x \triangleright \mathbf{0} \mid a \bullet P$$

These two reductions reflect non-determinism inherent in the situation, which can be readily interpreted, for instance, as a choice between the receipt of a message on channel b and a failure of location a. Such conflicts appear to us to be non problematic for (a) they reflect inherently non-deterministic situations that should be captured in a distributed process calculus, such as a potential occurrence of a failure, and (b) they involve pure messages and control behavior located at a higher level in the hierarchy of kells, which is consistent with a view of locations being organized in a hierarchy of control, and which can readily be implemented even in a ditributed setting (although at more expense than a simple asynchronous message communication, but that is only to be expected).

Another potential conflict situation in dK is exemplified by configuration D below:

$$D = a \bullet x \triangleright \mathbf{0} \mid a \bullet x \triangleright x \mid a \bullet P$$

which has the following two reductions:

$$D \to a \bullet x \triangleright \mathbf{0} \mid P \qquad D \to a \bullet x \triangleright x$$

Here again, we do not see this situation as problematic for the conflict takes place between two triggers placed at the same location. If the location is interpreted as a computing site, then the conflict is purely local and can be readily implemented as a mere scheduling decision. If the location is to be interpreted as a network, then the a triggers correspond to the modelling of some inherent non-determinism in the network behavior.

Interestingly, we do not use lose much in expressive power when going from the full Kell calculus to dK. All the encodings given in previous sections are actually encodings in dK. The subcalculus dK thus appears as an interesting basis for an implementable distributed calculus.

Second, we can envisage relying on type systems to enforce required safety properties such as the linearity of certain names to ensure the determinacy of routing as is done, in a similar higher-order context, in the M-calculus [25]. We leave the study of such type systems for future research. Third, and most importantly, note that the interaction behavior of a kell depends on the environment (i.e. surrounding kell) it is placed in. The Kell calculus, contrary to Mobile Ambients, allows the coexistence of different forms of interactions between kells, as well as different communication environments. In particular, one could think of a particular environment that would only allow the exchange of passive kells, thus enforcing the constraint that characterizes the dK subcalculus, and the creation and destruction of sites, i.e. kells corresponding to different spatial loci of computation. All subkells of such sites would then be deemed local to the site. Furthermore, messages exchanged between sites could be deemed to bear an address name, i.e. a name that would be unique among the sites and that would at any time be associated univocally with one site only, again a linearity property that could be enforced by a type system. This interpretation, which is strikingly similar to the distributed implementation of the DJoin calculus and of the M-calculus, allows a direct implementation in a distributed setting. None of the conflict situations described above exist between sites, and conflict situations involving non site kells can be resolved locally, within a site. Communication between sites can thus be mapped to standard point-to-point asynchronous communication, with more complex communication protocols needed only when there is kell mobility between sites.

4 Advanced patterns

We introduce in this section a more sophisticated pattern language for Kell calculus triggers. Patterns defined in this section provide more introspection capabilities than simple patterns. Such introspection capabilities can be leveraged for programming various forms of routing and control, by means of wrappers, interceptors, and the like. We illustrate this through encodings of the Distributed Join calculus and of the M-calculus.

$$
\begin{array}{rcl}
\xi & ::= & \rho' \mid \rho' :: \pi \mid \xi \mid \xi \\
\rho' & ::= & \rho \mid \rho^{\uparrow} \mid \rho_{\downarrow} \mid \rho_{l} \\
\rho & ::= & a \mid a * u \mid a * \xi \mid a * (u \mid \xi) \\
u & ::= & a \mid (a) \mid (x) \\
\pi & ::= & a = b \mid b \in \mathsf{T} \mid b \in \mathsf{D} \mid \neg\pi \mid \pi \wedge \pi \\
\mathsf{T} & ::= & \mathsf{K} \mid b \bullet \mathsf{T} \\
\mathsf{D} & ::= & \mathsf{P} \mid b \bullet \mathsf{D}
\end{array}
$$

Figure 11: Syntax of advanced patterns

4.1 Syntax and semantics

The syntax of advanced patterns is given in Figure 11. Advanced patterns are built on simple patterns extended with direction and with the ability to do some introspection on the content of the current kell. Intuitively, a pattern of the form $\rho' :: \pi$ matches kells which match the pattern ρ, provided that in the current evaluation context they match the indicated direction and that the predicate π is satisfied (note that the scope of variables appearing in pattern ρ extends to predicate π). We abbreviate $a \neq b$ the predicate $\neg(a = b)$, $a \notin \mathsf{T}$ the predicate $\neg(a \in \mathsf{T})$, $a \notin \mathsf{D}$ the predicate $\neg(a \in \mathsf{D})$, and $\pi_1 \vee \pi_2$ the predicate $\neg(\pi_1 \wedge \pi_2)$. The intuition behind the predicates given in Figure 11 is as follows:

- Predicate $a = b$ is true when name a is equal to name b.
- In a given context $\mathbf{C} \equiv c \bullet (\cdot \mid Q)$, predicate $a \in \mathsf{K}$ is true when there is a subkell in kell c that has name a, or when the predicate is true, recursively, of a subkell of c.
- In a given context $\mathbf{C} \equiv c \bullet (\cdot \mid Q)$, predicate $a \in b \bullet \mathsf{T}$ is true when there is a subkell b of c such that predicate $a \in \mathsf{T}$ is true of this kell.
- In a given context $\mathbf{C}$, a pattern of the form $\rho :: \neg\pi$ matches a kell which matches the ρ pattern provided the predicate π does not hold in the context. Likewise, a pattern of the form $\rho :: \pi_1 \wedge \pi_2$ matches a kell which matches the ρ pattern provided both predicates π_1 and π_2 hold in the context.

The formal semantics of advanced patterns is defined as follows. Matching for advanced patterns is defined as a conservative extension of that of simple patterns. More precisely, a context-dependent substitution θ is a standard substitution mapping name markers to Kell calculus processes that is extended to work in contexts $\mathbf{C}$ as defined below:

$$(\rho' :: \pi)\theta_{\mathbf{C}} = M \quad \equiv \quad \rho\theta = M \ \wedge \ \psi(M, \rho', \mathbf{C}) \ \wedge \ \phi(\pi\theta, M, \rho, \mathbf{C})$$

where ψ and ϕ are predicates defined inductively on all contexts of the form $\mathbf{C} = N \mid a \bullet (\cdot \mid Q)$ by:

$$\begin{array}{rcl}
\psi(M, \rho, \mathbf{C}) & = & \mathtt{true} \\
\psi(M, \rho^{\uparrow}, \mathbf{C}) & = & \mathbf{C} \equiv N \mid M \mid a \bullet (\cdot \mid Q) \\
\psi(M, \rho_{\downarrow}, \mathbf{C}) & = & \mathbf{C} \equiv N \mid a \bullet (\cdot \mid b \bullet (M \mid Q) \mid R) \\
\psi(M, \rho_{l}, \mathbf{C}) & = & \mathbf{C} \equiv N \mid a \bullet (\cdot \mid M \mid R)
\end{array}$$

$$\begin{array}{rcl}
\phi(b \in \mathsf{K}, M, \rho, \mathbf{C}) & = & \mathbf{C} \equiv N \mid a \bullet (\cdot \mid \mathbf{E}[b \bullet R]) \\
\phi(b \in c_1 \bullet \ldots c_n \bullet \mathsf{K}, M, \rho, \mathbf{C}) & = & \mathbf{C} \equiv N \mid a \bullet (\cdot \mid c_1 \bullet \ldots (c_n \bullet \mathbf{E}[b \bullet R] \mid Q_n) \mid \ldots \mid Q_1) \\
\phi(b \in \mathsf{P}, M, \rho, \mathbf{C}) & = & \mathbf{C} \equiv N \mid a \bullet (\cdot \mid \xi \triangleright R \mid Q) \ \wedge \ b \in \mathtt{fn}(\xi) \\
\phi(b \in c_1 \bullet \ldots c_n \bullet \mathsf{K}, M, \rho, \mathbf{C}) & = & \mathbf{C} \equiv N \mid a \bullet (\cdot \mid c_1 \bullet (\ldots c_n \bullet (\xi \triangleright R \mid Q_n) \mid \ldots \mid Q_1) \mid Q) \\
 & & \wedge\ b \in \mathtt{fn}(\xi) \\
\phi(a = b, M, \rho, \mathbf{C}) & = & (a = b) \\
\phi(\neg\pi, M, \rho, \mathbf{C}) & = & \neg\phi(\pi, M, \rho, \mathbf{C}) \\
\phi(\pi_1 \wedge \pi_2, M, \rho, \mathbf{C}) & = & \phi(\pi_1, M, \rho, \mathbf{C}) \wedge \phi(\pi_2, M, \rho, \mathbf{C})
\end{array}$$

The structural congruence relation on advanced patterns is defined by the following rules: the parallel operator, $|$, is associative and commutative; if two patterns ξ and ζ are equivalent ($\xi \equiv \zeta$), then so are $a * \xi$ and $a * \zeta$; if the two predicates π and π' are logically equivalent, then the patterns $\rho' :: \pi$ and $\rho' :: \pi'$ are equivalent; if two patterns differ only by α-conversion (taking into account bound identifiers), then they are equivalent. Simple patterns form the subset of advanced patterns which are deemed to be of sort S. All other advanced patterns (patterns with direction, and patterns with predicate) are deemed to be only of sort H.

4.2 Encoding the DJoin calculus and the M-calculus

Using advanced patterns allows straightforward encodings of the DJoin calculus and of the M-calculus. An encoding of the DJoin can be derived via the composition of the encoding of the Djoin in the M-calculus described in [25] and of the encoding of the M-calculus defined in this section. However, a more direct encoding can be obtained as follows. For any DJoin definition D, we note $\mathtt{df}(D)$ the set of names (channels and locations) it defines. The DJoin encoding is a function of a name that keeps track of the current DJoin location. It is defined by induction as follows, where we assume that m, mm, loc, collect, toGC, make, va, enter do not occur free in P, D:

$$\begin{array}{ll}
[\![a]\!]_b = a & [\![\mathbf{0}]\!]_b = \mathbf{0} \\
[\![\top]\!]_b = \mathbf{0} & [\![P \mid Q]\!]_b = [\![P]\!]_b \mid [\![Q]\!]_b \\
[\![\mathtt{go}\, a; P]\!]_b = \mathtt{va}\langle a, [\![P]\!]_a \rangle & [\![D, D']\!]_b = [\![D]\!]_b \mid [\![D']\!]_b \\
[\![a\langle n_1, \ldots, n_q \rangle]\!]_b = \mathtt{m}\langle b, a, n_1, \ldots, n_q \rangle & [\![D \,\mathtt{in}\, P]\!]_b = \nu\widetilde{n}.\, [\![D]\!]_b \mid [\![P]\!]_b \qquad \widetilde{n} = \mathtt{df}(D)
\end{array}$$

$$[\![n_1\widetilde{m_1} \mid \ldots \mid n_q\widetilde{m_q} \triangleright P]\!]_b = \mathtt{m}\langle b, n_1, \widetilde{(m_1)} \rangle \mid \ldots \mid \mathtt{m}\langle b, n_q, \widetilde{(m_q)} \rangle \diamond [\![P]\!]_b$$
$$[\![a[D : P]]\!]_b = a \bullet (\mathtt{DJ}(a) \mid \mathtt{loc} \bullet ([\![D]\!]_a \mid [\![P]\!]_a)) \mid \mathtt{DJEnv}$$

together with the following auxiliary definitions:

$$
\begin{array}{rcl}
\mathtt{DJ}(a) & = & \nu \mathtt{t}.\, \mathtt{t} \mid \mathtt{IR} \mid \mathtt{Go}(a,\mathtt{t}) \mid \mathtt{Enter}(a,\mathtt{t}) \\
\mathtt{IR} & = & \mathtt{m}\langle (a), x\rangle^{\uparrow} :: a \in \mathtt{K} \mid \mathtt{loc} \bullet p \diamond \mathtt{loc} \bullet (p \mid \mathtt{m}\langle a, x\rangle) \\
 & & \mid\ \mathtt{m}\langle (a), x\rangle_{\downarrow} :: a \notin \mathtt{K} \diamond \mathtt{mm}\langle a, x\rangle \\
\mathtt{Go}(a,\mathtt{t}) & = & \mathtt{t} \mid \mathtt{va}\langle (c), p\rangle_{\downarrow} :: c \notin \mathtt{K} \triangleright \\
 & & \quad (\mathtt{loc} \bullet q \triangleright \\
 & & \quad\ \nu k.\ \mathtt{toGC}\langle a, k\rangle \mid \mathtt{collect}\langle a, k, \mathtt{enter}\langle c, a, (q \mid p)\rangle\rangle) \\
\mathtt{Enter}(a,\mathtt{t}) & = & \mathtt{t} \mid \mathtt{enter}\langle a, (b), x\rangle^{\uparrow} \mid \mathtt{loc} \bullet p \diamond \\
 & & \nu k.\mathtt{make}\langle b, x, k\rangle \triangleright (k\langle y\rangle^{\uparrow} \triangleright \mathtt{t} \mid \mathtt{loc} \bullet (p \mid y)) \\
\mathtt{DJEnv} & = & \nu \mathtt{t}.(\mathtt{C}\,\mathtt{t}\,\mathtt{C}) \mid \mathtt{ER} \mid \mathtt{GC} \\
\mathtt{C} & = & \lambda \mathtt{t}\, f.\, \mathtt{Factory}(\mathtt{t}) \mid \mathtt{t}\langle f\rangle \\
\mathtt{Factory}(\mathtt{t}) & = & \mathtt{t}\langle f\rangle \mid \mathtt{make}\langle n, p, k\rangle_{\downarrow} \diamond k\langle \nu a.(f\, a\, f) \mid \mathtt{ER} \mid \mathtt{GC} \mid n \bullet (\mathtt{DJ}(n) \mid \mathtt{loc} \bullet p)\rangle \\
\mathtt{ER} & = & \mathtt{mm}\langle x\rangle_{\downarrow} \diamond \mathtt{m}\langle x\rangle \\
\mathtt{GC} & = & \mathtt{collect}\langle (a), (k), p\rangle \diamond (a \bullet (z \mid \mathtt{toGC}\langle a, k\rangle) \triangleright p)
\end{array}
$$

Some comments are in order. Note that the encoding of a DJoin locality takes the same general form as that of a Mobile Ambient: a locality a has a controlling process $\mathtt{DJ}(a)$, that implements the basic interaction protocol that governs a DJoin locality. The latter includes: routing messages on the basis of the target locality, implementing locality migration, by means of the $\mathtt{Go}(a)$ and $\mathtt{Enter}(a)$ processes. Note that the encoding given above is faithful to the DJoin semantics, since migration is only allowed if the target locality does not appear as a sublocality of the current locality.[5]

We can likewise define an encoding of the M-calculus. For simplicity, we consider some slight modifications on the syntax given in [25]. In particular, we consider that each resource name variable, locality name variable and process variable are properly distinguished. Also, we consider a call-by-name evaluation strategy for the λ-calculus constructs of the calculus, instead of a call-by-value one. As for Mobile Ambients and the DJoin calculus, we make use of the λ-abstraction and application constructs defined in Section 3.3. The encoding of the M-calculus in the Kell calculus is defined by induction as follows, where we assume that the names m, mm, mmm,mb, pm, pass, nil, make, collect, toGC do not occur free in V, V_i, P, Q, and where we note (x) for a variable that can be a (Kell calculus) name marker or a process marker, depending on the sort (resource or name variable, process variable) of the source variable in the translation.

$$
\begin{array}{ll}
[\![a]\!] = a & [\![\mathbf{0}]\!] = \mathbf{0} \\
[\![x]\!] = x & [\![()]\!] = \mathtt{nil} \\
[\![\nu a.P]\!] = \nu a.[\![P]\!] & [\![\nu r.P]\!] = \nu r.[\![P]\!] \\
[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!] & [\![\lambda x.P]\!] = \lambda x.[\![P]\!] \\
[\![PQ]\!] = [\![P]\!][\![Q]\!] & [\![r\langle V_1, \ldots, V_q\rangle]\!] = \mathtt{m}\langle r, [\![V_1]\!], \ldots, [\![V_q]\!]\rangle
\end{array}
$$

[5] This is not the case of the encoding of the DJoin calculus in the M-calculus defined in [25], which does not test for the presence of the target locality as a sublocality of the locality to be migrated. It is possible to faithfully encode the Djoin calculus in the M-calculus but at the cost of a more complex translation than the one reported in [25].

$$
\begin{array}{l}
[\![a.r\langle V_1,\ldots,V_q\rangle]\!] = \mathtt{mm}\langle a, r, [\![V_1]\!], \ldots, [\![V_q]\!]\rangle \\
[\![[a=b]P,Q]\!] = \nu l.\ l\langle(b)\rangle :: a = b \triangleright [\![P]\!] \mid l\langle(b)\rangle :: a \neq b \triangleright [\![Q]\!] \mid l\langle b\rangle \\
[\![r_1\widetilde{x_1} \mid \ldots \mid r_q\widetilde{x_q} \triangleright P]\!] = \mathtt{m}\langle r_1, \widetilde{(x_1)}\rangle \mid \ldots \mid \mathtt{m}\langle r_q, \widetilde{(x_q)}\rangle \diamond [\![P]\!] \\
[\![a(P)[Q]]\!] = a\bullet(\mathtt{M}(a) \mid \mathtt{mb}\bullet[\![P]\!] \mid \mathtt{pm}\bullet[\![Q]\!]) \mid \mathtt{MEnv} \\
[\![\mathtt{pass}\ V]\!] = \mathtt{pm}\bullet q \triangleright \mathtt{pass}\langle[\![V]\!], q\rangle
\end{array}
$$

together with the following auxiliary definitions:

$$
\begin{array}{rcl}
\mathtt{M}(a) & = & \mathtt{IR}(a) \mid \mathtt{Pass}(a) \\
\mathtt{Pass}(a) & = & \mathtt{mb}\bullet(p \mid \mathtt{pass}\langle x, q\rangle) \triangleright (\nu k.\ \mathtt{collect}\langle a, k, (x\,a\,p\,q)\rangle \mid \mathtt{toGC}\langle a, k\rangle) \\
\mathtt{MEnv} & = & \mathtt{ER} \mid \mathtt{GC} \\
\mathtt{ER} & = & \mathtt{mmm}\langle x\rangle_{\downarrow} \diamond \mathtt{mm}\langle x\rangle \\
\mathtt{GC} & = & \mathtt{collect}\langle(a),(k),p\rangle_{\downarrow} \diamond (a\bullet(z \mid \mathtt{toGC}\langle a, k\rangle) \triangleright p)
\end{array}
$$

Auxiliary process $\mathtt{IR}(a)$ provides a direct encoding of the M-calculus routing rules, as defined in [25]:

$$
\begin{array}{rcl}
\mathtt{IR}(a) & = & \mathtt{mm}\langle(b),(r),\widetilde{(x)}\rangle^{\uparrow} :: b = a \vee b \in \mathtt{mb}\bullet\mathsf{K} \vee b \in \mathtt{pm}\bullet\mathsf{K} \diamond \mathtt{m}\langle\mathtt{i}, a, r, \widetilde{x}\rangle \\
& & \mid \mathtt{mb}\bullet(z \mid \mathtt{mm}\langle a,(r),\widetilde{(x)}\rangle) \diamond \mathtt{mb}\bullet(z \mid \mathtt{m}\langle r, \widetilde{x}\rangle) \\
& & \mid \mathtt{pm}\bullet(z \mid \mathtt{mm}\langle a,(r),\widetilde{(x)}\rangle) \diamond \mathtt{pm}\bullet(z \mid \mathtt{m}\langle r, \widetilde{x}\rangle) \\
& & \mid (\mathtt{mb}\bullet(p \mid \mathtt{mm}\langle(b),(r),\widetilde{(x)}\rangle) :: b \neq a \wedge b \in \mathtt{pm}\bullet\mathsf{K} \wedge b \notin \mathtt{mb}\bullet\mathsf{K} \\
& & \quad \mid \mathtt{pm}\bullet q \diamond \\
& & \quad\ \mathtt{mb}\bullet p \mid \mathtt{pm}\bullet(q \mid \mathtt{mm}\langle b, r, \widetilde{x}\rangle)) \\
& & \mid (\mathtt{mb}\bullet(p \mid \mathtt{mm}\langle(b),(r),\widetilde{(x)}\rangle) :: b \neq a \wedge b \notin \mathtt{pm}\bullet\mathsf{K} \wedge b \notin \mathtt{mb}\bullet\mathsf{K} \diamond \\
& & \quad \mathtt{mb}\bullet p \mid \mathtt{mmm}\langle b, r, \widetilde{x}\rangle) \\
& & \mid (\mathtt{pm}\bullet(q \mid \mathtt{mm}\langle(b),(r),\widetilde{(x)}\rangle) :: b \neq a \wedge b \notin \mathtt{pm}\bullet\mathsf{K} \\
& & \quad \mid \mathtt{mb}\bullet p \diamond \\
& & \quad\ \mathtt{pm}\bullet q \mid \mathtt{mb}\bullet(p \mid \mathtt{m}\langle\mathsf{o}, b, r, \widetilde{x}\rangle)) \\
& & \mid (\mathtt{mb}\bullet(p \mid \mathtt{m}\langle r, \widetilde{(x)}\rangle) :: r \notin \mathtt{mb}\bullet\mathsf{P} \wedge r \in \mathtt{pm}\bullet\mathsf{P} \\
& & \quad \mid \mathtt{pm}\bullet q \diamond \\
& & \quad\ \mathtt{mb}\bullet p \mid \mathtt{pm}\bullet(q \mid \mathtt{m}\langle r, \widetilde{(x)}\rangle)) \\
& & \mid (\mathtt{pm}\bullet(q \mid \mathtt{m}\langle r, \widetilde{(x)}\rangle) :: r \notin \mathtt{pm}\bullet\mathsf{P} \wedge r \in \mathtt{mb}\bullet\mathsf{P} \\
& & \quad \mid \mathtt{mb}\bullet p \diamond \\
& & \quad\ \mathtt{pm}\bullet q \mid \mathtt{mb}\bullet(p \mid \mathtt{m}\langle r, \widetilde{(x)}\rangle))
\end{array}
$$

An M-calculus programmable locality $a(P)[Q]$ takes the form of a kell of the same name, with a controller process $\mathtt{M}(a)$ that embodies the basic routing rules of the M-calculus and implements the behavior of the `pass` operator. Controller process P and content process Q appear as active

subkells. Note that we use a different garbage collector, GC, that exploits a pattern of the form $a * (x \mid \xi)$. Note also that to obtain en encoding of the M-calculus in the subcalculus dK introduced in Section 3.7, it suffices to modify the handling of the `pass` construct thus:

$$[\![\texttt{pass}\ V]\!] = \texttt{pass}\langle [\![V]\!] \rangle$$
$$\texttt{Pass}(a) = \texttt{mb}\bullet(p \mid \texttt{pass}\langle x \rangle) \mid \texttt{pm}\bullet q \triangleright (\nu k.\, \texttt{collect}\langle a, k, (x\, a\, p\, q)\rangle \mid \texttt{toGC}\langle a, k\rangle)$$

4.3 Encoding the distributed interpretation

Interestingly, the distributed interpretation we have sketched at the end of Section 3.7 can be defined as a direct subcalculus of the Kell calculus. Its syntax is given below (top-level kells are called sites):

$$\begin{array}{rcl} C & ::= & \texttt{DKEnv} \mid S \mid C \mid S \\ S & ::= & a\bullet(\texttt{Router}(a) \mid \texttt{site}\bullet P) \end{array}$$

It semantics is given by the following definitions, where we assume that names m, mm, newS, stop do not occur free in P:

$$\begin{array}{rcl} \texttt{Router}(a) & = & \texttt{m}\langle a, x\rangle^{\uparrow} \diamond \texttt{m}\langle a, x\rangle \\ & & \mid \texttt{m}\langle (b), x\rangle_{\downarrow} :: b \neq a \diamond \texttt{mm}\langle b, x\rangle \\ \texttt{DKEnv} & = & \texttt{mm}\langle (a), x\rangle \diamond \texttt{m}\langle a, x\rangle \\ & & \mid \texttt{newS}\langle (a), (k), p\rangle \diamond \nu s.\, s\bullet(\texttt{Router}(s) \mid p) \mid \texttt{m}\langle a, k, s\rangle \\ & & \mid \texttt{stop}\langle (a), (k), (b)\rangle :: b \neq a \diamond (b\bullet z) \triangleright \texttt{m}\langle a, k, \texttt{ok}\rangle \\ & & \mid \texttt{stop}\langle (a), (k), (b)\rangle :: b = a \diamond (a\bullet z) \triangleright \mathbf{0} \end{array}$$

The Router process just moves messages in and out of a given site. Messages $\texttt{m}\langle a, P\rangle$ between sites bear an explicit target site a, together with a payload P. The environment DKEnv assists in the routing of messages, and provides for the creation and destruction of sites. Creation and destruction of sites are notified back to the originator of the newS and stop commands.

This interpretation requires that messages identify systematically the target site. This is very similar to what takes place in Dπ or in the lower Nomadic Pict calculus. If one want to recover a more location transparent communication between sites, we could instead expect the routing mechanism to maintain the knowledge of an association between potential message targets and hosting sites. This again is easy to capture in the Kell calculus, yielding a calulculus with location transparent communication based on uniquely located channels. This is in turn very similar to what takes place in the distributed implementation of the DJoin, or in the upper Nomadic Pict calculus. This new interpretation can be obtained just by modifying the Router process thus:

$$\begin{array}{rcl} \texttt{Router}(a) & = & \texttt{m}\langle (b), x\rangle^{\uparrow} :: b \in \texttt{site}\bullet\texttt{K} \diamond \texttt{m}\langle b, x\rangle \\ & & \mid \texttt{m}\langle (b), x\rangle_{\downarrow} :: b \notin \texttt{site}\bullet\texttt{K} \diamond \texttt{mm}\langle b, x\rangle \end{array}$$

$$
\begin{array}{rcl}
P & ::= & \mathbf{0} \mid a \mid x \mid a\langle u_1, \ldots, u_n\rangle \triangleright P \mid a \bullet u \triangleright P \mid \nu a.P \mid (P \mid P) \mid a * P \\
N & ::= & a \mid a\langle P_1, \ldots, P_n\rangle \mid a \bullet P \\
M & ::= & a \mid a\langle P_1, \ldots, P_n\rangle \\
u & ::= & (a) \mid (x) \\
* & ::= & \bullet \mid \circ
\end{array}
$$

Figure 12: Syntax of the μdK Calculus

To be clear, messages in this interpretation always take the form $\mathtt{m}\langle a, x\rangle$ where a designates the target kell, a kell which is not a site. A target kell can be understood for instance as an object, as illustrated in Section 3.4, or as an M-calculus location, as defined above. The routing scheme defined above is determinate only if one can ensure the unicity of target kell names, e.g. either by static scoping as in the DJoin, or by means of an appropriate type system as in the M-calculus.

5 Parting notes

We gather in this section various remarks and additional discussions concerning the Kell calculus and some of its sub-calculi.

5.1 The μdK calculus

In the previous section, we have introduced a relatively sophisticated pattern language, that allowed some degree of introspection on the structure of a kell. What happens if we go in the reverse direction, i.e. rather than complexify, if we simplify the simple pattern language introduced in Section 3?

One way to simplify it is to retain the possibility to match only one message at a time, and to allow as messages the equivalent of messages in the higher-order polyadic asynchronous π-calculus, i.e. messages of the form $k\langle P_1, \ldots, P_n\rangle$. The resulting sub-calculus we call the μdK calculus for we believe it merits to be investigated further. For future reference, we give here the syntax of the μdK calculus as well as the reduction rules that result from the choices made. The full syntax of the μdK calculus is given in Figure 12.

The reduction semantics of the μdK calculus is given by the same structural congruence as the Kell calculus, without rule S.TRIG, which is not necessary (it is covered by S.α). The reduction relation is given by the rules in Figure 13. They correspond to the rules R.CONTEXT, R.EQUIV in Figure 5, and to the rules IN, OUT, and BETA in Figure 6. The expression $\xi\theta = M$ means that: if $\xi = a\langle u_1, \ldots, u_n\rangle$, then $M = a\langle P_1, \ldots, P_n\rangle$ and, for all $i \in \{1, \ldots, n\}$, if $u_i = (a_i)$, then P_i is a name b_i, if $u_j = (x_j)$, then P_j is an arbitrary (μdK calculus) process. The expression $\xi\theta = N$ means that: if $\xi = a\langle u_1, \ldots, u_n\rangle$, then N must be as M above; if $\xi = a \bullet (b)$, then $N = a \bullet c$, where c is some name; and if $\xi = a \bullet (x)$, then $N = a \bullet P$, where P is an arbitrary μdK calculus process. The substitution θ is just the substitution $\{b_i/a_i, P_j/x_j\}$.

$$\frac{\xi\theta \equiv M}{a \bullet (\xi \triangleright P \mid Q) \mid M \to a \bullet (P\theta \mid Q)}\ [\textsc{In}] \qquad \frac{\xi\theta \equiv M}{\xi \triangleright P \mid a \bullet (M \mid Q) \to P\theta \mid a \bullet Q}\ [\textsc{Out}]$$

$$\frac{\xi\theta \equiv N}{\xi \triangleright P \mid N \to P\theta}\ [\textsc{Beta}] \qquad \frac{P \to Q}{\mathbf{E}[P] \to \mathbf{E}[Q]}\ [\textsc{R.Context}]$$

$$\frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q}\ [\textsc{R.Equiv}]$$

Figure 13: Reduction relation of the μdK Calculus

It is interesting to note that one can encode the Kell calculus with simple patterns in this calculus: it suffices to emulate the matching on multiple messages by multiple successive receptions and the other features of the pattern matching in simple patterns by multiple embedded receptions. This in turn means that this calculus is already expressive enough to provide a faithful encoding of different notions of locations, including ambient-like notions. Also, this calculus is a subcalculus of the dK calculus, introduced Section 3.7. This means that the μdK calculus can readily support a distributed interpretation and, coupled with an appropriate type system to ensure the linearity of (certain) kell names, would be amenable to an efficient distributed implementation.

5.2 Why two boxes ?

It would seem that the box operator $\circ$ is redundant since $a \bullet a \triangleright P$ prevents the execution of P and still make it available for further manipulation through the a handle. The following reductions illustrate this:

$$a \bullet (a \triangleright P) \mid (a \bullet x \triangleright x \mid a) \to a \triangleright P \mid a \to P$$

However, at a minimum, an explicit construction of the form $a \circ P$ provides a natural distinction between pure messages and active locations, a distinction which we have exploited in the dK subcalculus to avoid conflict situations arising with active kells. If we had only the $\bullet$ box operator, we would lose that distinction. The only way to recover it would be through the means of an extended pattern language which would allow matching on processes of the form $a \triangleright P$. But then we would uncover a whole new set of potential conflict situations, since $a \triangleright P$, like $a \bullet P$, would play a dual role of message and receiver. Overall then, it seems that having the $\circ$ operator actually simplifies the calculus and provides the necessary basis for its distributed interpretation.

5.3 Why components ?

The title of this paper announces a calculus for higher-order distributed components. So far, however, we have not mentioned components. What gives ? A first element of answer can actually be found in Section 3.4, where we showed that the Kell calculus (and in fact the dK subcalculus) provided an easy intepretation of objects as active kells. Beyond objects, an active kell may have several

subkells, and can provide different forms of control over these subkells. As we saw in Section 3.5, a kell can interrupt a subkell and later resume it execution. The encoding of the M-calculus in Section 4.2 illustrates the possibility for a kell to act as interceptor for messages ultimately destined to subkells. This suggests that kells with subkells look very much like composite components with subcomponents. Since communication between subkells of a kell k must be mediated by kell k, we can implement different forms of communication paths or connectors [13] between subkells. One can also control explicitly within kell k the establishment and release of such connectors, providing the basis for a dynamic component-based structure. For instance, a receiving port named a of a component can readily be represented as a receiver of a kell, able to match messages of the form $a\langle . \rangle$. Likewise, a sending port named a can be equated with the possibility for a kell to emit messages of the form $a\langle . \rangle$. Given two kells k_1 and k_2, the first one with a sending port a and the second one with a receiving port b, a simple connector between ports a and b can be defined as $C(a,b) = a\langle x \rangle \diamond b\langle x \rangle$. If some control needs to be exercized on a connector, then it is possible to use some shared lock $\mathtt{t}$ as in $C(\mathtt{t}, a, b) = \mathtt{t} \mid a\langle x \rangle \diamond \mathtt{t} \mid b\langle x \rangle$. If the lock is consumed by the controller process, then the connector ceases functioning. Alternatively, one could define the connector as a kell $C(a,b) = c \bullet (a\langle x \rangle \diamond b\langle x \rangle)$, and use trivial connectors $a\langle x \rangle \mid \mathtt{t} \diamond a\langle x \rangle \mid \mathtt{t}$ and $b\langle x \rangle \mid \mathtt{t} \diamond b\langle x \rangle \mid \mathtt{t}$ to ensure connectivity between ports and connectors. Deleting or suspending the connector c can then be done easily.

As another illustration of the possibility to use the Kell calculus as a basic composition language, we can define a direct encoding of the $\pi\mathcal{L}$[6], calculus, which serves as a basis for the Piccola composition language [19]. The $\pi\mathcal{L}$ calculus is an asynchronous π-calculus, which handles forms F instead of names. Forms are records of fields $l = V$, where V is value, which can be either a name a or a field selection x_l, where x is a form variable.

$$
\begin{array}{ll}
[\![a]\!]_k = k\langle a \rangle & [\![x_l]\!]_k = x \mid (l \circ x \triangleright k\langle x \rangle) \\
[\![\mathcal{E}]\!] = \mathbf{0} & [\![F\langle l = V \rangle]\!] = [\![F]\!] \mid l \circ [\![V]\!] \\
[\![Fx]\!] = [\![F]\!] \mid x & [\![\mathbf{0}]\!] = \mathbf{0} \\
[\![A \mid B]\!] = [\![A]\!] \mid [\![B]\!] & [\![\nu a.A]\!] = \nu a.[\![A]\!] \\
[\![V(x).A]\!] = \nu k.[\![V]\!]_k \mid (k\langle (a) \rangle \triangleright a\langle x \rangle \triangleright [\![A]\!]) & [\![\overline{V}(F).A]\!] = \nu k.[\![V]\!]_k \mid (k\langle (a) \rangle \triangleright a\langle [\![F]\!] \rangle) \\
[\![!V(x).A]\!] = \nu k.[\![V]\!]_k \mid (k\langle (a) \rangle \diamond a\langle x \rangle \triangleright [\![A]\!]) &
\end{array}
$$

We can therefore develop, using the Kell calculus as a basis, the same type of constructs that have been developed in Piccola, such as generic wrappers, complex and active forms, composition scripts, etc. In fact, the above encoding of $\pi\mathcal{L}$ calculus provides a direct way to extend Piccola to a distributed setting by just adding the active kell construct of the Kell calculus.

6 Conclusion

We have introduced in this paper a new process calculus, called the Kell calculus, and demonstrated, through various encodings, its expressive power. In particular, we have shown that it faithfully

[6] For simplicity, we make one slight alteration to the $\pi\mathcal{L}$ calculus defined in [19]: forms can have multiple fields with the same label, as in the $\pi\mathcal{L}$ calculus, but we do not guarantee that a field selection will return the rightmost one.

captures the semantics of locations in several process calculi such as ambient calculi, the DJoin calculus and the M-calculus. We have discussed how suitable restrictions of the calculus yield sub-calculi which can be implemented efficiently in a distributed environment, lending some weight to the belief that the Kell calculus could constitute a suitable basis for distributed computing. Further work remains to confirm that belief, through. First, we would like to apply the same techniques used in the M-calculus to derive a type system for the Kell calculus which can ensure the linearity of chosen names. We have seen that this is a crucial point for the distributed interpretation of the calculus. Second, we need to investigate the bisimulation semantics of the calculus. Indeed, apart from the context rules, the rules of reduction of the calculus are essentially local rules, i.e. rules which involve only a single location. We believe this feature of the calculus can allow us to directly leverage the results obtained in the past decade on the bisimulation semantics of process algebras with localities [8]. Third, we believe a crucial feature for a distributed process calculus is to be able to deal with overlapping locations, i.e. locations that may share processes. This is necessary to deal properly with the physical and logical aspects of locations in a distributed setting. How overlapping locations can be defined remains as a challenge, though.

Acknowledgments

Many thanks to G. Boudol for a fruitful discussion on the different features of the Kell calculus, and to members of the Mikado project for interesting comments on early versions of the calculus presented in this report.

References

[1] M. Abadi and L. Cardelli. *A theory of objects*. Springer, 1996.

[2] R. Amadio. An asynchronous model of locality, failure, and process mobility. Technical report, INRIA Research Report RR-3109, INRIA Sophia-Antipolis, France, 1997.

[3] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science, vol. 96*, 1992.

[4] G. Boudol. The π-calculus in direct style. *Higher-Order and Symbolic Computation, vol.11*, 1998.

[5] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, 2001.

[6] L. Cardelli. Types for mobile ambients. In *Proceedings 26th Annual ACM Symposium on Principles of Programming Languages (POPL)*, 1999.

[7] L. Cardelli and A. Gordon. Mobile ambients. In *Foundations of Software Science and Computational Structures, M. Nivat (Ed.), Lecture Notes in Computer Science, Vol. 1378*. Springer Verlag, 1998.

[8] I. Castellani. Process algebras with localities. In *Handbook of Process Algebra, J. Bergstra, A. Ponse and S. Smolka (eds)*. Elsevier, 2001.

[9] C. Fournet. *The Join-Calculus*. PhD thesis, Ecole Polytechnique, Palaiseau, France, 1998.

[10] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *In proceedings 23rd ACM Symposium on Principles of Programming Languages (POPL)*, 1996.

[11] C. Fournet, G. Gonthier, J.J. Levy, L. Maranget, and D. Remy. A calculus of mobile agents. In *In Proceedings 7th International Conference on Concurrency Theory (CONCUR '96), Lecture Notes in Computer Science 1119*. Springer Verlag, 1996.

[12] C. Fournet, J.J. Levy, and A. Schmitt. An asynchronous distributed implementation of mobile ambients. In *Proceedings of the International IFIP Conference TCS 2000, Sendai, Japan, Lecture Notes in Computer Science 1872*. Springer, 2000.

[13] D. Garlan, R. Monroe, and D. Wile. *Acme: Architectural Description of Component-Based Systems*, chapter 3. In [15], 2000.

[14] A. Gordon and P. Hankin. A Concurrent Object Calculus. In *3rd International Workshop on High-Level Concurrent Languages, ENTCS*, 1998.

[15] G. Leavens and M. Sitaraman (eds). *Foundations of Component-Based Systems*. Cambridge University Press, 2000.

[16] L. Leth and B. Thomsen. Some facile chemistry. *Formal Aspects of Computing Vol.7, No 3*, 1995.

[17] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, 2000.

[18] L. Lopes, F. Silva, A. Figueira, and V. Vasconcelos. DiTyCO: An Experiment in Code Mobility from the Realm of Process Calculi. In *Proceedings 5th Mobile Object Systems Workshop (MOS'99)*, 1999.

[19] M. Lumpe, F. Achermann, and O. Nierstrasz. *A Formal Language for Composition*, chapter 4. In [15], 2000.

[20] M. Merro and M. Hennessy. Bisimulation congruences in safe ambients. In *29th ACM Symposium on Principles of Programming Languages (POPL), Portland, Oregon, 16-18 January*, 2002.

[21] R. Milner. Calculi for interaction. *Acta Informatica, Vol.33, No 8*, 1996.

[22] R. Milner. *Communicating and mobile systems : the π-calculus*. Cambridge University Press, 1999.

[23] D. Sangiorgi and A. Valente. A Distributed Abstract Machine for Safe Ambients. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 2001.

[24] D. Sangiorgi and S. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[25] A. Schmitt and J.B. Stefani. The M-calculus: A Higher-Order Distributed Process Calculus. In *Proceedings 30th Annual ACM Symposium on Principles of Programming Languages (POPL)*, 2003.

[26] D. Teller, P. Zimmer, and D. Hirschkoff. Using Ambients to Control Resources. In *to appear in Proceedings CONCUR 02*, 2002.

[27] B. Thomsen. A Theory of Higher Order Communicating Systems. *Information and Computation, Vol. 116, No 1*, 1995.

[28] J. Vitek and G. Castagna. Towards a calculus of secure mobile computations. In *Proceedings Workshop on Internet Programming Languages, Chicago, Illinois, USA, Lecture Notes in Computer Science 1686, Springer*, 1998.

[29] P. Wojciechowski and P. Sewell. Nomadic Pict: Language and Infrastructure. *IEEE Concurrency, vol. 8, no 2*, 2000.

[30] N. Yoshida and M. Hennessy. Subtyping and locality in distributed higher-order processes. In *Proceedings CONCUR 99, Lecture Notes in Computer Science no 1664*. Springer, 1999.

[31] S. Dal Zilio. *Le calcul bleu : types et objets*. PhD thesis, U. of Nice-Sophia Antipolis, France, 1999.

A Proofs

We gather in this section the proofs of theorems appearing in the main text, together with auxiliary lemmas.

Lemma A.1 (Substitution Lemma) *For all θ, $\mathbf{C}$, P, Q, if $P \equiv Q$ and $\theta_{\mathbf{C}} n \cap bn(P \mid Q) = \emptyset$, then $P\theta_{\mathbf{C}} \equiv Q\theta_{\mathbf{C}}$.*

Proof: A simple induction on the derivation of $P \equiv Q$. □

Lemma A.2 *If $P \xrightarrow{\mathbf{K} : A} P'$, then we have $fn(\mathbf{K}) \subseteq fn(P)$ and $bn(A) \subseteq bn(P)$.*

Proof: A simple induction on the derivation of $P \xrightarrow{\mathbf{K} : A} P'$. □

We define recursively the operation $\{.\}$ on actions by: recursively: $\{\epsilon\} = \epsilon$, $\{\tau\} = \tau$, $\{\alpha\} = \alpha$, $\{\beta\} = \beta$, $\{\langle A \rangle\} = \{A\}$, $\{A \mid B\} = \{A\} \mid \{B\}$. Note that if $A \neq \epsilon, \tau$, then $\{\lceil A \rceil\}$ is a Kell calculus process. When $\{A\} \equiv \beta \mid \prod_{i \in J} \alpha_i$, $\beta = (\xi, \theta_{\mathbf{C}})$, we say that β or θ *occurs in* A. By the following lemma, an action β can occur only at most once in a given action A obtained in a derivation $P \xrightarrow{\mathbf{K} : A} P'$. By convention, we set $\prod_{i \in J} A_i = \epsilon$ if $J = \emptyset$.

Lemma A.3 *If $P \xrightarrow{\mathbf{K} : A} P'$, then we have (i) $A = \tau$, or (ii) $\{A\} \equiv \prod_{i \in J} \alpha_i$ or (iii) $\{A\} \equiv \beta \mid \prod_{i \in J} \alpha_i$. Furthermore, in case (i) and (ii) $\cdot$ does not occur in $\mathbf{K}$, and in case (iii) $\cdot$ occurs exactly once in $\mathbf{K}$.*

Proof: By induction on the derivation of $P \xrightarrow{\mathbf{K} : A} P'$. The key induction step is the case when $P \xrightarrow{\mathbf{K} : A} P'$ is derived through the use of rule L.PAR. In this case, we have $P = R_1 \mid R_2$, $R_i \xrightarrow{\mathbf{K}_i : A_i} R_i'$, $A = A_1 \mid A_2$, $\mathbf{K} = \mathbf{K}_1 \mid \mathbf{K}_2$, with the constraint that $\cdot$ occurs at most once in $\mathbf{K}$, and if one of the A_i is equal to τ, then the other must be ϵ. Because of this constraint, and using the induction hypothesis for A_1 and A_2, we then have five possible cases: (1) A_1 verifies (ii) and A_2 verifies (ii), in which case A verifies (ii); (2) A_1 verifies (ii) and A_2 verifies (iii), in which case A verifies (iii); (3) A_1 verifies (iii) and A_2 verifies (ii), in which case A verifies (iii); (4) A_1 verifies (i) and $A_2 = \epsilon$, i.e. A_2 verifies (ii) with $J = \emptyset$, and A verifies (i). (5) A_2 verifies (i) and $A_1 = \epsilon$, i.e. A_2 verifies (ii) with $J = \emptyset$, and A verifies (i). Hence A verifies (i), (ii) or (iii), as required. □

Lemma A.4 *If $P \xrightarrow{\mathbf{K} : A} P'$, then $fn(A) \subseteq fn(P) \cup fn(\beta)$ and $fn(P') \subseteq fn(P) \cup bn(A) \cup fn(\beta)$ if β occurs in A; otherwise, $fn(A) \subseteq fn(P)$ and $fn(P') \subseteq fn(P) \cup bn(A)$.*

Proof: By induction on the derivation of $P \xrightarrow{\mathbf{K} : A} P'$. By convention, we set $\mathtt{fn}(\beta) = \emptyset$ if β does not occur in A.

- L.NULL. In this case, we have $A = \epsilon$, and $P' = \mathbf{K} = P$. Hence $\mathtt{fn}(A) = \emptyset \subseteq \mathtt{fn}(P)$ and $\mathtt{fn}(P') = \mathtt{fn}(P)$, as required.
- L.ACT and L.PASS. In these cases, we have $A = P$, $\mathbf{K} = P$ and $P' = \mathbf{0}$. Hence $\mathtt{fn}(A) = \mathtt{fn}(P)$ and $\mathtt{fn}(P') = \emptyset \subseteq \mathtt{fn}(P)$, as required.

- L.TRIG. In this case, we have $P = \xi \triangleright R$, $A = (\xi, \theta_{\mathbf{C}}) = \beta$, $\mathbf{K} = \cdot$, $P' = R\theta_{\mathbf{C}}$, $\mathtt{fn}(P) = \mathtt{fn}(\xi) \cup (\mathtt{fn}(R) \setminus \mathtt{bn}(\xi))$. Hence $\mathtt{fn}(A) = \mathtt{fn}(\beta)$, and $\mathtt{fn}(P') = \mathtt{fn}(R) \setminus \mathtt{bn}(\xi) \cup \mathtt{fn}(\theta_{\mathbf{C}}.\mathtt{cosupp}) \subseteq \mathtt{fn}(P) \cup \mathtt{fn}(\beta)$, as required.

- L.LOC. In this case, we have $P = a \bullet R$, $R \xrightarrow{\mathbf{K}' : A'} R'$, $A = \langle A' \rangle$, $\mathbf{K} = a \bullet \mathbf{K}'$, $P' = a \bullet R'$, $\mathtt{fn}(P) = \{a\} \cup \mathtt{fn}(R)$, $\mathtt{fn}(P') = \{a\} \cup \mathtt{fn}(R')$. By induction hypothesis, we have $\mathtt{fn}(A') \subseteq \mathtt{fn}(R) \cup \mathtt{fn}(\beta)$. Hence $\mathtt{fn}(A) = \mathtt{fn}(A') \subseteq \mathtt{fn}(P) \cup \mathtt{fn}(\beta)$, as required. Also by induction hypothesis, we have $\mathtt{fn}(R') \subseteq \mathtt{fn}(R) \cup \mathtt{bn}(A') \cup \mathtt{fn}(\beta)$. Hence $\mathtt{fn}(P') \subseteq \mathtt{fn}(R) \cup \{a\} \cup \mathtt{bn}(A) \cup \mathtt{fn}(\beta) = \mathtt{fn}(P) \cup \mathtt{bn}(A) \cup \mathtt{fn}(\beta)$, as required.

- L.NU.NF. In this case, we have $P = \nu a.R$, $R \xrightarrow{\mathbf{K}' : A'} R'$, $A = A'$, $\mathbf{K} = \overline{\nu} a.\mathbf{K}'$, $P' = \nu a.R'$, $\mathtt{fn}(P) = \mathtt{fn}(R) \setminus \{a\}$, $\mathtt{ln}(P) = \mathtt{ln}(R)$, $\mathtt{fn}(P') = \mathtt{fn}(R') \setminus \{a\}$. By induction hypothesis, we have $\mathtt{fn}(A') \subseteq \mathtt{fn}(R) \cup \mathtt{fn}(\beta)$ and $\mathtt{fn}(R') \subseteq \mathtt{fn}(R) \cup \mathtt{fn}(\beta) \cup \mathtt{bn}(A')$. Hence $\mathtt{fn}(A) \subseteq \mathtt{fn}(P) \cup \mathtt{fn}(\beta)$ and $\mathtt{fn}(P') \subseteq \mathtt{fn}(P) \cup \mathtt{fn}(\beta) \cup \mathtt{bn}(A)$, as required.

- L.NU.F. In this case, we have $P = \nu a.R$, $R \xrightarrow{\mathbf{K}' : A'} R'$, $A = \overline{\nu} a.A'$, $\mathbf{K} = \overline{\nu} a.\mathbf{K}'$, $P' = \nu a.R'$, $\Gamma = \Gamma' \cup \{a\}$, $\mathtt{fn}(P) = \mathtt{fn}(R) \setminus \{a\}$, $\mathtt{fn}(A) = \mathtt{fn}(A') \setminus \{a\}$, $\mathtt{bn}(A) = \mathtt{bn}(A') \cup \{a\}$, $\mathtt{fn}(P') = \mathtt{fn}(R') \setminus \{a\}$. By induction hypothesis, we have $\mathtt{fn}(A') \subseteq \mathtt{fn}(R) \cup \mathtt{fn}(\beta)$ and $\mathtt{fn}(R') \subseteq \mathtt{fn}(R) \cup \mathtt{fn}(\beta) \cup \mathtt{bn}(A')$. Hence $\mathtt{fn}(A) \subseteq (\mathtt{fn}(R) \cup \mathtt{fn}(\beta)) \setminus \{a\} \subseteq \mathtt{fn}(R) \setminus \{a\} \cup \mathtt{fn}(\beta) = \mathtt{fn}(P) \cup \mathtt{fn}(\beta)$ and $\mathtt{fn}(P') = \mathtt{fn}(R') \setminus \{a\} \subseteq\subseteq \mathtt{fn}(R) \setminus \{a\} \cup \mathtt{bn}(A') \cup \mathtt{fn}(\beta) \subseteq \mathit{fn}(()P) \cup \mathtt{bn}(A) \cup \mathtt{fn}(\beta)$, as required.

- L.PAR. In this case, we have $P = R_1 \mid R_2$, $R_i \xrightarrow{\mathbf{K}_i : A_i} R_i$, $A = A_1 \mid A_2$, $\mathbf{K} = \mathbf{K}_1 \mid \mathbf{K}_2$, $P' = R_1' \mid R_2'$, $\mathtt{fn}(P) = \mathtt{fn}(R_1) \cup \mathtt{fn}(R_2)$, $\mathtt{fn}(A) = \mathtt{fn}(A_1) \cup \mathtt{fn}(A_2)$, $\mathtt{bn}(A) = \mathtt{bn}(A_1) \cup \mathtt{bn}(A_2)$, $\mathtt{fn}(P') = \mathtt{fn}(R_1') \cup \mathtt{fn}(R_2')$. By induction hypothesis, we have $\mathtt{fn}(A_i) \subseteq \mathtt{fn}(R_i) \cup \mathtt{fn}(\beta_i)$ and $\mathtt{fn}(R_i') \subseteq \mathtt{fn}(R_i) \cup \cup \mathtt{bn}(A_i \cup \mathtt{fn}(\beta_i)$. By Lemma A.3, there is at most one β occurring in A. So at least one of $\mathtt{fn}(\beta_i) = \emptyset$. Let β be the potentially occurring one. Hence, we have $\mathtt{fn}(A) \subseteq \mathtt{fn}(R_1) \cup \mathtt{fn}(R_2) \cup \mathtt{fn}(\beta) = \mathtt{fn}(P) \cup \mathtt{fn}(\beta)$ and $\mathtt{fn}(P') \subseteq \mathtt{fn}(R_1) \cup \mathtt{fn}(R_2) \cup \mathtt{bn}(A_1 \cup \mathtt{bn}(A_2) \cup \mathtt{fn}(\beta) = \mathtt{fn}(P) \cup \mathtt{bn}(A) \cup \mathtt{fn}(\beta)$, as required.

- L.RED.S. In this case, we have $P \xrightarrow{\mathbf{K}' : A'} P''$, $A = \tau$, $\mathbf{K} = P$, $\widetilde{c} = \mathtt{bn}(A')$ $P' = \nu \widetilde{c}.P''$, $\mathtt{fn}(P') = \mathtt{fn}(P'') \setminus \widetilde{c}$, $A' = A_1 \mid \beta \mid A_2$, $\beta = (\xi, \theta_{\mathbf{K}'})$, $\xi\theta_{\mathbf{K}'} = [A_1 \mid A_2]$. By induction hypothesis, we have $\mathtt{fn}(A') \subseteq \mathtt{fn}(P) \cup \mathtt{fn}(\beta)$ and $\mathtt{fn}(P'') \subseteq \mathtt{fn}(P) \cup \mathtt{bn}(A') \cup \mathtt{fn}(\beta)$. By Lemma A.3, we know that no β action occurs in A_i, hence we have, by induction hypothesis since A_i must have appeared in an earlier derivation step involving subterms of P, $\mathtt{fn}(A_i) \subseteq \mathtt{fn}(P) \cup \widetilde{c}$. By definition $\mathtt{fn}(\beta) = \mathtt{fn}(\xi) \cup \mathtt{fn}(\theta_{\mathbf{K}}.\mathtt{cosupp})$. But $\xi\theta_{\mathbf{K}} = [A_1 \mid A_2]$, hence $\mathtt{fn}(\theta_{\mathbf{K}}.\mathtt{cosupp}) = \mathtt{fn}(A_1 \mid A_2)$ and $\mathtt{fn}(\beta) \subseteq \mathtt{fn}(P) \cup \widetilde{c}$. As a result, we have $\mathtt{fn}(A) = \emptyset \subseteq \mathtt{fn}(P)$, and $\mathtt{fn}(P') \subseteq (\mathtt{fn}(P) \cup \widetilde{c} \cup \mathtt{fn}(\beta)) \setminus \widetilde{c} \subseteq \mathtt{fn}(P)$, as required.

- L.RED.H. This case is handled as the case L.RED.S above.

□

Proof of Theorem 1: The proof proceeds by induction on the depth of the inference of $P \xrightarrow{\mathbf{K} : A} P'$ and by induction on the depth of the derivation of $P \equiv Q$ by means of the structural congruence rules.

We first consider the case when $P \equiv Q$ has been obtained by the application of a single structural congruence rule, and we consider the last rule that has been applied in the derivation of $P \xrightarrow{\mathbf{K} : A} P'$. In the

different cases below, several subcases can be dispatched immediately. Because we consider only derivations of $P \equiv Q$ which only involve a single inference step, rule S.CONTEXT is not applicable. Also, if $P \equiv Q$ has been derived using rule S.α, then by definition of the transition relation (which is defined up to α-conversion), if $P \xrightarrow{\mathbf{K} : A} P'$ and $P =_\alpha Q$, then $Q \xrightarrow{\mathbf{K} : A} P'$. Finally, if $P \equiv Q$ has been obtained through S.PAR.NIL, i.e. $Q = P \mid \mathbf{0}$, then by rule L.PAR.L we get $Q \xrightarrow{\mathbf{K} : A} P' \mid \mathbf{0}$, and we have found $Q' = P' \mid \mathbf{0} \equiv P'$, $\mathbf{K}' = \mathbf{K}$ and $B = A$, as required. In the sequel, we do not consider these subcases any more.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.NULL. In this case we have $\mathbf{K} = P$, $A = \epsilon$ and $P' = P$. We have by L.NULL $Q \xrightarrow{Q : \epsilon} Q$. Hence we have found $Q' = Q \equiv P = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, and $B = \epsilon = A$, as required.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.ACT. In this case, we have $P = a \bullet R$, $P' = \mathbf{0}$ and $\mathbf{K} = A = P$. Since $P \equiv Q$, we have $a \bullet R \equiv Q$. This could only have been derived by rule S.NU.KELL (apart from the three rules which have been considered above). Thus, we have $P = a \bullet \nu b.S$ and $Q = \nu b.a \bullet S$, with $b \neq a$. There are now two possibilities: either (i) $b \in \mathtt{fn}(S)$ or (ii) $b \notin \mathtt{fn}(S)$.

In case (i), we have in fact $b \in \mathtt{fon}(a \bullet S)$; hence we can apply rule L.ACT followed by L.NU.F to get: $Q \xrightarrow{Q : \overline{\nu} b.a \bullet S} \mathbf{0}$. Since $b \neq a$, we have $\overline{\nu} b.a \bullet S \equiv a \bullet \overline{\nu} b.S$: we have found $B = \overline{\nu} b.a \bullet S \equiv a \bullet \overline{\nu} b.S = A$, $\mathbf{K}' = Q \equiv \mathbf{K} = P$ and $Q' = \mathbf{0} = P'$, as required.

In case (ii), we can apply rule L.ACT followed by L.NU.NF to get: $Q \xrightarrow{Q : a \bullet S} \nu b.\mathbf{0}$. But we have $\nu b.\mathbf{0} \equiv \mathbf{0}$ and , since $a \neq b$, we have $a \bullet \overline{\nu} b.S \equiv \overline{\nu} b.a \bullet S \equiv \overline{\nu} b.a \bullet S \mid \epsilon \equiv a \bullet S \mid \overline{\nu} b.\epsilon \equiv a \bullet S$, hence we have found $B = a \bullet S \equiv a \bullet \overline{\nu} b.S = A$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$ and $Q' = \nu b.\mathbf{0} \equiv \mathbf{0} = P'$ as required.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.PASS. In this case, we have $P = a \circ R$, $P' = \mathbf{0}$ and $\mathbf{K} = A = P$. Since $P \equiv Q$, we have $a \circ R \equiv Q$. This could only have been derived by rules: (a) S.α, (b) S.CONTEXT, or (c) S.PAR.NIL, which have all been considered above.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.TRIG. In this case, we have $P = \xi \triangleright R$, $\mathbf{K} = \cdot$, $A = (\xi, \theta_{\mathbf{C}})$, and $P' = P\theta_{\mathbf{C}}$. But then $P = \xi \triangleright R \equiv Q$ could only have obtained through S.TRIG. Thus, we have $Q = \zeta \triangleright R$. But then by L.TRIG, we have $Q \xrightarrow{\cdot : (\zeta, \theta_{\mathbf{C}})} R\theta_{\mathbf{C}}$. Hence we have obtained $Q' = R\theta_{\mathbf{C}} = P'$, $\mathbf{K}' = \cdot = \mathbf{K}$, and $B = (\zeta, \theta_{\mathbf{C}}) \equiv (\xi, \theta_{\mathbf{C}}) = A$, as required.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.NU.NF. In this case we have $P = \nu a.R$, $R \xrightarrow{\mathbf{K}_1 : A} R'$ with $a \notin \mathtt{fn}(A)$, $P' = \nu a.R'$, and $\mathbf{K} = \overline{\nu} a.\mathbf{K}_1$. Since $P \equiv Q$ we have $Q \equiv \nu a.R$. This could only have been derived by rules: (a) S.NIL, (b) S.NU.KELL, (c) S.NU.COMM, (d) S.NU.PAR.

- In case (a), we have $R = \mathbf{0}$, $P = \nu a.\mathbf{0}$, and $Q = \mathbf{0}$. But then $R \xrightarrow{\mathbf{K}_1 : A} R'$ could only have been derived by L.NULL, hence $\mathbf{K}_1 = \mathbf{0}$, $A = \epsilon$, $R' = \mathbf{0}$, and $\mathbf{K} = \overline{\nu} a.\mathbf{0} \equiv \nu a.\mathbf{0} = P$, $P' = \nu a.\mathbf{0} = P$. By L.NULL we have $Q \xrightarrow{Q : \epsilon} Q$, hence we have found $Q' = Q \equiv P = P'$, $\mathbf{K}' = Q \equiv P \equiv \mathbf{K}$, and $B = \epsilon = A$, as required.

- In case (b), we have $R = b \bullet S$, with $b \neq a$, and $Q = b \bullet \nu a.S$. Now $R \xrightarrow{\mathbf{K}_1 : A} R'$ could only have been derived through rules (i) L.ACT, (ii) L.LOC, (iii) L.RED.S, (iv) L.RED.H, (v) L.NULL.

  - In case (i) we have $b \bullet S \xrightarrow{b \bullet S : b \bullet S} \mathbf{0}$. Hence $\mathbf{K}_1 = b \bullet S$, $A = b \bullet S$, and $R' = \mathbf{0}$. By rule L.ACT one gets: $b \bullet \nu a.S \xrightarrow{b \bullet \nu a.S : b \bullet \nu a.S} \mathbf{0}$. Since $a \notin \mathtt{fn}(A)$, we have $a \notin \mathtt{fn}(S)$,

hence we have: $\nu a.b \bullet S \equiv b \bullet S$. Hence we have found :$Q' = R' = \mathbf{0} \equiv \nu a.R' = P'$, $B = b \bullet \nu a.S \equiv \nu a.b \bullet S \equiv b \bullet S = A$, and $\mathbf{K}' = b \bullet \nu a.Sequiv b \bullet S \equiv \overline{\nu} a.b \bullet S = \mathbf{K}$, as required.

– In case (ii), we have $b \bullet S \xrightarrow{b \bullet \mathbf{K}_2 : \langle A' \rangle} b \bullet S'$, with $S \xrightarrow{\mathbf{K}_2 : A'} S'$, $A = \langle A' \rangle$, $\mathbf{K}_1 = b \bullet \mathbf{K}_2$, and $R' = b \bullet S'$. Since $a \notin \mathtt{fn}(A)$, then $a \notin \mathtt{fn}(A')$, and by rule L.NU.NF one gets: $\nu a.S \xrightarrow{\overline{\nu} a.\mathbf{K}_2 : A'} \nu a.S'$. Applying rule L.LOC yields: $b \bullet \nu a.S \xrightarrow{b \bullet \overline{\nu} a.\mathbf{K}_2 : \langle A' \rangle} b \bullet \nu a.S'$. Since $b \neq a$, we have: $b \bullet \nu a.S' \equiv \nu a.b \bullet S'$, and $b \bullet \overline{\nu} a.\mathbf{K}_2 \equiv \overline{\nu} a.b \bullet \mathbf{K}_2$. Hence we have found $Q' = b \bullet \nu a.S' \equiv \nu a.b \bullet S' = \nu a.R' = P'$, $B = \langle A' \rangle = A$, and $\mathbf{K}' = b \bullet \overline{\nu} a.\mathbf{K}_2 \equiv \overline{\nu} a.b \bullet \mathbf{K}_2 = \overline{\nu} a.\mathbf{K}_1 = \mathbf{K}$, as required.

– In case (iii), we must have $\mathbf{K}_1 = R$, $A = \tau$ and $R \xrightarrow{\mathbf{K}_2 : A_2} R''$ with $\lceil A_2 \rceil = A_1' \mid (\xi, \theta_{\mathbf{C}}) \mid A_2'$. But we have $R = b \bullet S$, hence $R \xrightarrow{\mathbf{K}_2 : A_2} R''$ could only have been obtained via L.LOC, hence A_2 shold be such that $A_2 = \langle A_3 \rangle$, a contradiction. Hence this case is void.

– In case (iv) we have $A = \tau$, $\mathbf{K}_1 = b \bullet S = R$, $R' = \nu \widetilde{c}.R''$, with $R \xrightarrow{\mathbf{K}_2 : A_2} R''$, $\widetilde{c} = \mathtt{bn}(A_2)$, $\lceil A_2 \rceil \equiv A_3' \mid \langle A_1' \mid (\xi, \theta_{\mathbf{C}}) \mid A_2' \rangle \mid A_4'$, $\mathbf{C} = \lceil \mathbf{K}_2 \rceil$, $\xi\theta_{\mathbf{C}} = A_3' \mid [A_1' \mid A_2'] \mid A_4'$, $|A_3'|, |A_4'| = 0$, $|A_1'|, |A_2'| \leq 1$. Now, $R = b \bullet S \xrightarrow{\mathbf{K}_2 : A_2} R''$ could only have been obtained through rule L.LOC (rule L.ACT does not apply because of the form of action A_2). Hence we must have: $A_3' = A_4' = \epsilon$, $S \xrightarrow{\mathbf{K}_3 : A_3} S'$, $\mathbf{K}_2 = b \bullet \mathbf{K}_3$, $A_2 = \langle A_3 \rangle$, and $R'' = b \bullet S'$. We now have two cases to consider: (1) $a \notin \mathtt{fn}(A_3)$ and (2) $a \in \mathtt{fn}(A_3)$.

In case (1), we can apply L.NU.NF to get: $\nu a.S \xrightarrow{\overline{\nu} a.\mathbf{K}_3 : A_3} \nu a.S'$. Now, by L.LOC, we get: $b \bullet \nu a.S \xrightarrow{b \bullet \overline{\nu} a.\mathbf{K}_3 : \langle A_3 \rangle} b \bullet \nu a.S'$. The conditions of premises of rule L.RED.H still apply, in particular since $\langle A_3 \rangle = A_2$, and since $\lceil \mathbf{K}_2 \rceil = \lceil b \bullet \mathbf{K}_3 \rceil = \lceil b \bullet \overline{\nu} a.\mathbf{K}_3 \rceil$. Hence, by L.RED.H we get:

$$b \bullet \nu a.S \xrightarrow{b \bullet \nu a.S : \tau} \nu \widetilde{c}.b \bullet \nu a.S'$$

Now, since a is bound in P, and since the transition rules are defi ned up to α-conversion, it is always possible to choose a such that $a \notin \widetilde{c}$. Hence we have found $Q' = \nu \widetilde{c}.b \bullet \nu a.S' \equiv \nu a.\nu \widetilde{c}.b \bullet S' = \nu a.\nu \widetilde{c}.R'' = \nu a.R' = P'$, $\mathbf{K}' = b \bullet \nu a.S \equiv \overline{\nu} a.b \bullet S = \overline{\nu} a.\mathbf{K}_1 = \mathbf{K}$, and $B = \tau = A$, as required.

In case (2) we can consider that $a \mathtt{fon}(A_3)$ for if that were not the case, then by Lemma A.4, we would have $a \notin \mathtt{fn}(S)$, a contradiction with the fact that $\xi\theta_{\mathbf{C}} = [A_1' \mid A_2']$. We can then apply L.NU.F to get: $\nu a.S \xrightarrow{\overline{\nu} a.\mathbf{K}_3 : \overline{\nu} a.A_3} S'$. Now, applying rule L.LOC, we get:

$$b \bullet \nu a.S \xrightarrow{b \bullet \overline{\nu} a.\mathbf{K}_3 : \langle \overline{\nu} a.A_3 \rangle} b \bullet S'$$

Since $\lceil \langle \overline{\nu} a.A_3 \rangle \rceil = \lceil \langle A_3 \rangle \rceil = \lceil A_2 \rceil$, and $\lceil \mathbf{K}_2 \rceil = \lceil b \bullet \mathbf{K}_3 \rceil = \lceil b \bullet \overline{\nu} a.\mathbf{K}_3 \rceil$, we can apply L.RED.H to get:

$$b \bullet \nu a.S \xrightarrow{b \bullet \nu a.S : \tau} \nu a.\nu \widetilde{c}.b \bullet S'$$

Hence we have found $\mathbf{K}' = b \bullet \nu a.S \equiv \overline{\nu} a.b \bullet S = \overline{\nu} a.\mathbf{K}_1 = \mathbf{K}$, $Q' = \nu a.\nu \widetilde{c}.b \bullet S' = \nu a.\nu \widetilde{c}.R'' = \nu a.R' = P'$, and $B = \tau = A$, as required.

  - In case (v), we have $\mathbf{K}_1 = R$, $A = \epsilon$, and $R' = R$. Now, by L.NULL, we have $b \bullet \nu a.S \xrightarrow{b \bullet \nu a.S : \epsilon} b \bullet \nu a.S$, hence we have found $Q' = Q \equiv P = \nu a.b \bullet S = \nu a.R' = P'$, $\mathbf{K}' = Q \equiv P = \nu a.b \bullet S \equiv \overline{\nu} a.b \bullet S = \overline{\nu} a.R = \overline{\nu} a.\mathbf{K}_1 = \mathbf{K}$, and $B = \epsilon = A$, as required.

- In case (c), we have $R = \nu b.S$, $Q = \nu b.\nu a.S$ and $b \neq a$. Now, we could only have had $R \xrightarrow{\mathbf{K}_1 : A} R'$ through one of the following rules: (i) L.NU.NF, (ii) L.NU.F, (iii) L.RED.S, (iv) L.RED.H, (v) L.NULL.

  - In case (i), we have $\nu b.S \xrightarrow{\overline{\nu} b.\mathbf{K}_2 : A} \nu b.S'$, with $R' = \nu b.S'$, $S \xrightarrow{\mathbf{K}_2 : A} S'$, $\mathbf{K}_1 = \overline{\nu} b.\mathbf{K}_2$, and $b \notin \mathtt{fn}(A)$. Since $a \notin \mathtt{fn}(A)$, by rule L.NU.NF we get: $\nu a.S \xrightarrow{\overline{\nu} a.\mathbf{K}_2 : A} \nu a.S'$, and since $b \notin \mathtt{fn}(A)$, applying again rule L.NUNF we get: $\nu b.\nu a.S \xrightarrow{\overline{\nu} b.\overline{\nu} a.\mathbf{K}_2 : A} \nu b.\nu a.S'$. Since $b \neq a$ we have $\nu b.\nu a.S' \equiv \nu a.\nu b.S'$, and $\overline{\nu} b.\overline{\nu} a.\mathbf{K}_2 \equiv \overline{\nu} a.\overline{\nu} b.\mathbf{K}_2$. Hence we have found $Q' = \nu b.\nu a.S' \equiv \nu a.\nu b.S' = \nu a.R' = P'$, $B = A$, and $\mathbf{K}' = \overline{\nu} b.\overline{\nu} a.\mathbf{K}_2 \equiv \overline{\nu} a.\overline{\nu} b.\mathbf{K}_2 = \overline{\nu} a.\mathbf{K}_1 = \mathbf{K}$, as required.
  - In case (ii), we have $\nu b.S \xrightarrow{\overline{\nu} b.\mathbf{K}_2 : \overline{\nu} b.A_1} R'$, with $\mathbf{K}_1 = \overline{\nu} b.\mathbf{K}_2$ and $A = \overline{\nu} b.A_1$, with $b \in \mathtt{fn}(A_1)$ and $S \xrightarrow{\mathbf{K}_2 : A_1} R'$. Since $a \notin \mathtt{fn}(A)$ and $a \neq b$, then $a \notin \mathtt{fn}(A_1)$ and by rule L.NU.NF we get: $\nu a.S \xrightarrow{\overline{\nu} a.\mathbf{K}_2 : A_1} \nu a.R'$. Applying rule L.NU.F we get:

    $$\nu b.\nu a.S \xrightarrow{\overline{\nu} b.\overline{\nu} a.\mathbf{K}_2 : \overline{\nu} b.A_1} \nu a.R'$$

    Hence we have found $\mathbf{K}' = \overline{\nu} b.\overline{\nu} a.\mathbf{K}_2 \equiv \overline{\nu} a.\overline{\nu} b.\mathbf{K}_2 = \overline{\nu} a.\mathbf{K}_1 = \mathbf{K}$, $B = \overline{\nu} b.A_1 = A$ and $Q' = \nu a.R' = P'$, as required.
  - In case (iii), we have $R \xrightarrow{R : \tau} R'$ with $\mathbf{K}_1 = R$, $A = \tau$, $R' = \nu \widetilde{c}.R''$, $R \xrightarrow{\mathbf{K}_2 : A_1} R''$, $\widetilde{c} = \mathtt{bn}(A_1)$, $\lceil A_1 \rceil = A_1' \mid (\xi, \theta_{\mathbf{C}}) \mid A_2'$, $\mathbf{C} = \lceil \mathbf{K}_2 \rceil$, $\xi\theta_{\mathbf{C}} = [A_1' \mid A_2']$. Since $R = \nu b.S$, we could only have had $R = \nu b.S \xrightarrow{\mathbf{K}_2 : A_1} R''$ through either (1) rule L.NU.NF or (2) rule L.NU.F.

    In case (1), we have $\mathbf{K}_2 = \overline{\nu} b.\mathbf{K}_3$, $R'' = \nu b.S'$, $b \notin \mathtt{fn}(A_1)$ and $S \xrightarrow{\mathbf{K}_3 : A_1} S'$. We have $\lceil A_1 \rceil = A_1' \mid (\xi, \theta_{\mathbf{C}}) \mid A_2'$, and $\mathbf{C} = \lceil \mathbf{K}_2 \rceil = \lceil \overline{\nu} b.\mathbf{K}_3 \rceil = ceil\mathbf{K}_3$, hence we can apply L.RED.S to get: $S \xrightarrow{S : \tau} \nu \widetilde{c}.S'$. Applying L.NU.NF twice we get: $\nu b.\nu a.S \xrightarrow{\overline{\nu} b.\overline{\nu} a.S : \tau} \nu b.\nu a.\nu \widetilde{c}.S'$. Hence we have found: $\mathbf{K}' = \overline{\nu} b.\overline{\nu} a.S \equiv \overline{\nu} a.\overline{\nu} b.S = \overline{\nu} a.R = \mathbf{K}$, $B = \tau = A$, and $Q' = \nu b.\nu a.\nu \widetilde{c}.S' \equiv \nu a.\nu \widetilde{c}.\nu b.S' = \nu a.\nu \widetilde{c}.R'' = \nu a.R' = P'$, as required.

    In case (2), we have $\mathbf{K}_2 = \overline{\nu} b.\mathbf{K}_3$, $A_1 = \overline{\nu} b.A_2$, $b \in \mathtt{fn}(A_2)$, and $S \xrightarrow{\mathbf{K}_3 : A_2} R''$. Since $b \in \mathtt{bn}(A_1)$, we have $b \in \widetilde{c}$. We have $\lceil A_2 \rceil = \lceil \overline{\nu} a.A_2 \rceil = \lceil A_1 \rceil = A_1' \mid (\xi, \theta_{\mathbf{C}}) \mid A_2'$, and $\mathbf{C} = \lceil \mathbf{K}_2 \rceil = \lceil \overline{\nu} b.\mathbf{K}_3 \rceil = ceil\mathbf{K}_3$, hence we can apply L.RED.S with $\widetilde{c'} = \mathtt{bn}(A_2) = \widetilde{c} \setminus \{b\}$, to get: $S \xrightarrow{S : \tau} \nu \widetilde{c'}.R''$. Applying rule L.NU.NF twice we get: $\nu b.\nu a.S \xrightarrow{\overline{\nu} b.\overline{\nu} a.S : \tau} \nu b.\nu a.\nu \widetilde{c'}.R''$. Hence we have found: $Q' = \nu b.\nu a.\nu \widetilde{c'}.R'' \equiv \nu a.\nu \widetilde{c}.R'' = \nu a.R' = P'$, $\mathbf{K}' = \overline{\nu} b.\overline{\nu} a.S \equiv \overline{\nu} a.\overline{\nu} b.S = \overline{\nu} a.R = \mathbf{K}$, $B = \tau = A$, as required.
  - In case (iv), we reason exactly as in case (iii) above.

– In case (v), we have $\mathbf{K}_1 = R$, $A = \epsilon$, $R' = R$, hence $P' = \nu a.R = P$, and $\mathbf{K} = \overline{\nu} a.R \equiv P$. By L.NULL, we have $Q \xrightarrow{Q : \epsilon} Q$, hence we have found $Q' = Q \equiv P = P'$, $\mathbf{K}' = Q \equiv P \equiv \mathbf{K}$, and $B = \epsilon = A$, as required.

- In case (d), we have $P = \nu a.R$, $R = S \mid T$, $R \xrightarrow{\mathbf{K}_1 : A} R'$ with $a \notin \mathsf{fn}(A)$, $P' = \nu a.R'$, $\mathbf{K} = \overline{\nu} a.\mathbf{K}_1$, $Q = (\nu a.S) \mid T$ with $a \notin \mathsf{fn}(T)$. Now, $R \xrightarrow{\mathbf{K}_1 : A} R'$ could only have been derived via one of the following rules: (i) L.PAR, (ii) L.RED.S, (iii) L.RED.H, (iv) L.NULL.

  – In case (i), we have $S \xrightarrow{\mathbf{K}_2 : A_2} S'$, $T \xrightarrow{\mathbf{K}_3 : A_3} T'$, $\mathbf{K}_1 = \mathbf{K}_2 \mid \mathbf{K}_3$, $A = A_2 \mid A_3$, $R' = S' \mid T'$, $\mathsf{bn}(A_2) \cap \mathsf{bn}(A_3) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{fn}(T) = \emptyset$, $\mathsf{bn}(A_3) \cap \mathsf{fn}(S) = \emptyset$, and the conditions that if one of the A_2, A_3 is τ, then the other must be ϵ. Since $a \notin \mathsf{fn}(A)$, we have $a \notin \mathsf{fn}(A_2) \cup \mathsf{fn}(A_3)$, hence by rule L.NU.NF: $\nu a.S \xrightarrow{\overline{\nu} a.\mathbf{K}_2 : A_2} \nu a.S'$.Since $\mathsf{bn}(A_3) \cap \mathsf{fn}(\nu a.S) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{bn}(A_3) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{fn}(T) = \emptyset$, we can apply L.PAR to get:

$$(\nu a.S) \mid T \xrightarrow{(\overline{\nu} a.\mathbf{K}_2) \mid \mathbf{K}_3 : A_2 \mid A_3} (\nu a.S') \mid T'$$

  Since $a \notin \mathsf{fn}(T)$, then $a \notin \mathsf{fn}(\mathbf{K}_3)$ by Lemma A.2. Also, since $a \notin \mathsf{fn}(A_3)$ and $a \notin \mathsf{fn}(T)$, then by Lemma A.4 we have that $a \notin \mathsf{fn}(T')$ unless $a \in \mathsf{bn}(T)$. In the latter case, since the transition rules are given up to α-conversion, it is possible to α-convert T so that $a \notin \mathsf{bn}(T)$. Hence we have $a \notin \mathsf{fn}(\mathbf{K}_3)$ and $a \notin \mathsf{fn}(T')$. But then we have found $Q' = (\nu a.S') \mid T' \equiv \nu a.S' \mid T' = \nu a.R' = P'$, $\mathbf{K}' = (\overline{\nu} a.\mathbf{K}_2) \mid \mathbf{K}_3 \equiv \overline{\nu} a.\mathbf{K}_2 \mid \mathbf{K}_3 = \nu a.\mathbf{K}_1 = \mathbf{K}$, and $B = A_2 \mid A_3 = A$, as required.

  – In case (ii), we have $\mathbf{K}_1 = R$, $A = \tau$, $R' = \nu c.R''$, $R \xrightarrow{\mathbf{K}'_1 : A'_1} R''$, $\lceil A'_1 \rceil = A'_2 \mid (\xi, \theta_{\mathbf{C}}) \mid A'_2$, $\mathbf{C} = \lceil \mathbf{K}'_1 \rceil$, $\xi\theta_{\mathbf{C}} = [A'_2 \mid A'_3]$. Now, $R \xrightarrow{\mathbf{K}'_1 : A'_1} R''$ could only have been obtained by rule L.PAR, hence we have $S \xrightarrow{\mathbf{K}_2 : A_2} S'$, $T \xrightarrow{\mathbf{K}_3 : A_3} T'$, $\mathbf{K}'_1 = \mathbf{K}_2 \mid \mathbf{K}_3$, $A'_1 = A_2 \mid A_3$, $R'' = S' \mid T'$, $\mathsf{bn}(A_2) \cap \mathsf{bn}(A_3) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{fn}(T) = \emptyset$ $\mathsf{bn}(A_3) \cap \mathsf{fn}(S) = \emptyset$, and the constraint that if one A_2, A_3 equals τ, then the other must be ϵ. We now have to consider to subcases: (1) $a \notin \mathsf{fn}(A_2)$ and (2) $a \in \mathsf{fn}(A_2)$.

  In case (1), we can apply rule L.NU.NF to get: $\nu a.S \xrightarrow{\overline{\nu} a.\mathbf{K}_2 : A_2} \nu a.S'$. We have $\mathsf{bn}(A_3) \cap \mathsf{fn}(\overline{\nu} a.A_2) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{bn}(A_3) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{fn}(T) = \emptyset$, thus we can apply L.PAR to get:

$$(\nu a.S) \mid T \xrightarrow{(\overline{\nu} a.\mathbf{K}_2) \mid \mathbf{K}_3 : A_2 \mid A_3} (\nu a.S') \mid T'$$

  Now, we have $\lceil A_2 \mid A_3 \rceil = \lceil A'_1 \rceil$, $\mathbf{C} = \lceil \mathbf{K}'_1 \rceil = \lceil \mathbf{K}_2 \mid \mathbf{K}_3 \rceil = \lceil (\overline{\nu} a.\mathbf{K}_2) \mid \mathbf{K}_3 \rceil$, hence by L.RED.S we get:

$$(\nu a.S) \mid T \xrightarrow{(\nu a.S) \mid T : \tau} \nu \widetilde{c}.(\nu a.S') \mid T'$$

  Hence we have found $Q' = \nu \widetilde{c}.(\nu a.S') \mid T' \equiv \nu a.\nu \widetilde{c}.S' \mid T' = \nu a.R' = P'$, $B = \tau = A$, and $\mathbf{K}' = (\nu a.S) \mid T \equiv \overline{\nu} a.S \mid T = \overline{\nu} a.R = \overline{\nu} a.\mathbf{K}_1 = \mathbf{K}$, as required.

  In case (2), we can consider that $a \in \mathsf{fon}(A_2)$, for otherwise by Lemma A.4 we would have $a \notin \mathsf{fn}(S)$ and we would then have that $a \in \mathsf{fn}(\beta) = \mathsf{fn}(\xi, \theta_{\mathbf{C}})$; but since $\xi\theta_{\mathbf{C}}) = [A'_2 \mid A'_3]$, we

would have $a \in \mathsf{fn}(T)$, a contradiction. So we can apply L.NU.F to get: $\nu a.S \xrightarrow{\overline{\nu}a.\mathbf{K}_2 : \overline{\nu}a.A_2} S'$. We have $a \notin \mathsf{fn}(T)$ and $a \notin \mathsf{fn}(A_3)$, hence we have: $\mathsf{bn}(\overline{\nu}a.A_2) \cap \mathsf{bn}(A_3) = \emptyset$, $\mathsf{bn}(\nu a.A_2) \cap \mathsf{fn}(T) = \emptyset$, $\mathsf{bn}(A_3) \cap \mathsf{fn}(\nu a.S) = \emptyset$, hence by rule L.PAR we get:

$$(\nu a.S) \mid T \xrightarrow{(\overline{\nu}a.\mathbf{K}_2) \mid \mathbf{K}_3 : (\overline{\nu}a.A_2) \mid A_3} S' \mid T'$$

We have $\lceil \overline{\nu}a.A_2 \mid A_3 \rceil = \lceil A_2 \mid A_3 \rceil = \lceil A_1' \rceil$, $\mathbf{C} = \lceil \mathbf{K}_1' \rceil = \lceil \mathbf{K}_2 \mid \mathbf{K}_3 \rceil = \lceil \overline{\nu}a.\mathbf{K}_2 \mid \mathbf{K}_3 \rceil$, hence by L.RED.S:

$$(\nu a.S) \mid T \xrightarrow{(\nu a.S) \mid T : \tau} \nu a.\nu \widetilde{c}.S' \mid T'$$

Hence we have found $Q' = \nu a.\nu \widetilde{c}.S' \mid T' = \nu a.R' = P'$, $B = \tau = A$, and $\mathbf{K}' = (\nu a.S) \mid T \equiv \overline{\nu}a.S \mid T = \overline{\nu}a.R = \overline{\nu}a.\mathbf{K}_1 = \mathbf{K}$, as required.

- In case (iii), we reason exactly as in case (ii) above.
- tem In case (iv), we have $\mathbf{K}_1 = R$, $A = \epsilon$, $R' = R$, hence $P' = \nu a.R = P$, and $\mathbf{K} = \overline{\nu}a.R \equiv P$. By L.NULL, we have $Q \xrightarrow{Q : \epsilon} Q$, hence we have found $Q' = Q \equiv P = P'$, $\mathbf{K}' = Q \equiv P \equiv \mathbf{K}$, and $B = \epsilon = A$, as required.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.NU.F. In this case, we have $P = \nu a.R$, with $R \xrightarrow{\mathbf{K}_1 : A_1} P'$, and $a \in \mathsf{fon}(A_1)$, $A = \overline{\nu}a.A_1$, $\mathbf{K} = \overline{\nu}a.\mathbf{K}_1$. Since $P \equiv Q$, we have $Q \equiv \nu a.R$, and this could only have been derived via rules: (a)S.NIL, (b) S.NU.KELL, (c) S.NU.COMM, (d) S.NU.PAR.

- In case (a), we have $R = \mathbf{0}$ and $Q = \mathbf{0}$. Since $a \in \mathsf{fon}(A_1)$, $A_1 \neq \epsilon$, hence this case is void since there is no way to derive a transition for R except through rule L.NULL.
- In case (b), we have $R = b \bullet S$ with $b \neq a$ and $Q = b \bullet \nu a.S$. But then $R \xrightarrow{\mathbf{K}_1 : A_1} P'$ could only have been derived through (i) L.ACT, (ii) L.LOC, (iii) L.RED.H (rule L.RED.S cannot have been applied for the same reason as in subcase (b) of case L.NU.NF).
  - In case (i), we have $\mathbf{K}_1 = b \bullet S$, $A_1 = b \bullet S$, and $P' = \mathbf{0}$. But by L.ACT we have:

    $$b \bullet \nu a.S \xrightarrow{b \bullet \nu a.S : b \bullet \nu a.S} \mathbf{0}$$

    Hence we have found $Q' = \mathbf{0} = P'$, $\mathbf{K}' = b \bullet \nu a.S \equiv \overline{\nu}a.b \bullet S = \overline{\nu}a.\mathbf{K}_1 = \mathbf{K}$, and $B = b \bullet \nu a.S \equiv \overline{\nu}a.b \bullet S = \overline{\nu}a.A_1 = A$, as required.
  - In case (ii), we have $P' = b \bullet S'$, $S \xrightarrow{\mathbf{K}_2 : A_2} S'$, $\mathbf{K}_1 = b \bullet \mathbf{K}_2$, $A_1 = \langle A_2 \rangle$. Since $a \notin \mathsf{fn}(A_1)$, then $a \notin \mathsf{fn}(A_2)$, hence by L.NU.F: $\nu a.S \xrightarrow{\overline{\nu}a.\mathbf{K}_2 : \overline{\nu}a.A_2} S'$. Then, by L.LOC: $b \bullet \nu a.S \xrightarrow{b \bullet \nu a.\mathbf{K}_2 : \langle \overline{\nu}a.A_2 \rangle} b \bullet S'$. Hence we have found $Q' = b \bullet S' = P'$, $\mathbf{K}' = b \bullet \overline{\nu}a.\mathbf{K}_2 \equiv \overline{\nu}a.b \bullet \mathbf{K}_2 = \overline{\nu}a.\mathbf{K}_1 = \mathbf{K}$, and $B = \langle \overline{\nu}a.A_2 \rangle = \overline{\nu}a.\langle A_2 \rangle = \overline{\nu}a.A_1 = A$, as required.
  - In case (iii), we have $A_1 = \tau$, but then we have a contradiction for $a \in \mathsf{fon}(A_1)$, hence this case is void.
- In case (c), we have $R = \nu b.S$, with $b \neq a$, and $Q = \nu b.\nu a.S$. Now, we could only have had $R = \nu b.S \xrightarrow{\mathbf{K}_1 : A_1} P'$ through (i) L.NU.NF, (ii) L.NU.F, (iii) L.RED.S, (iv) L.RED.H.

- In case (i), we have $S \xrightarrow{\mathbf{K}_2 : A_2} S'$, with $\mathbf{K}_1 = \overline{\nu}b.\mathbf{K}_2$, $A_1 = A_2$, $P' = \nu b.S'$, $b \notin \mathtt{fn}(A_2)$. Since $a \in \mathtt{fon}(A_1)$, then by L.NU.F we get: $\nu a.S \xrightarrow{\overline{\nu}a.\mathbf{K}_2 : \overline{\nu}a.A_2} S'$. Now by L.NU.NF we get: $\nu b.\nu a.S \xrightarrow{\overline{\nu}b.\overline{\nu}a.\mathbf{K}_2 : \nu a.A_2} \nu b.S'$. Hence we have found $Q' = \nu b.S' = P'$, $\mathbf{K}' = \overline{\nu}b.\overline{\nu}a.\mathbf{K}_2 \equiv \overline{\nu}a.\overline{\nu}b.\mathbf{K}_2 = \overline{\nu}a.\mathbf{K}_1 = \mathbf{K}$, and $B = \overline{\nu}a.A_2 = \overline{\nu}a.A_1 = A$, as required.
- In case (ii), we have $S \xrightarrow{\mathbf{K}_2 : A_2} S'$, with $\mathbf{K}_1 = \overline{\nu}b.\mathbf{K}_2$, $A_1 = \overline{\nu}b.A_2$, $P' = S'$, $b \in \mathtt{fon}(A_2)$. Since $a \in \mathtt{fon}(A_1)$, then by L.NU.F we get: $\nu a.S \xrightarrow{\overline{\nu}a.\mathbf{K}_2 : \overline{\nu}a.A_2} S'$. Now by L.NU.F we get: $\nu b.\nu a.S \xrightarrow{\overline{\nu}b.\overline{\nu}a.\mathbf{K}_2 : \overline{\nu}b.\overline{\nu}a.A_2} S'$. Hence we have found $Q' = S' = P'$, $\mathbf{K}' = \overline{\nu}b.\overline{\nu}a.\mathbf{K}_2 \equiv \overline{\nu}a.\overline{\nu}b.\mathbf{K}_2 = \overline{\nu}a.\mathbf{K}_1 = \mathbf{K}$, and $B = \overline{\nu}b.\overline{\nu}a.A_2 \equiv \overline{\nu}a.\overline{\nu}b.A_2 = \overline{\nu}a.A_1 = A$, as required.
- In case (iii), we should have $\mathbf{K}_1 = R = \nu b.S$, $A_1 = \tau$, but since $a \in \mathtt{fon}(A_1)$, this case is void.
- In case (iv), we reason as in case (iii) above.

- In case (d), we have $R = S \mid T$ and $Q = (\nu a.S) \mid T$, $a \notin \mathtt{fn}(T)$. Now $R \xrightarrow{\mathbf{K}_1 : A_1} P'$ could only have been derived through rules (i) L.PAR, (ii) L.RED.S, or (iii) L.RED.H.
  - In case (i), we have $S \xrightarrow{\mathbf{K}_2 : A_2} S'$, $T \xrightarrow{\mathbf{K}_3 : A_3} T'$, $P' = S' \mid T'$, $\mathbf{K}_1 = \mathbf{K}_2 \mid \mathbf{K}_3$, $A_1 = A_2 \mid A_3$, $\mathtt{bn}(A_2) \cap \mathtt{bn}(A_3) = \emptyset$, $\mathtt{bn}(A_2) \cap \mathtt{fn}(T) = \emptyset$, $\mathtt{bn}(A_3) \cap \mathtt{fn}(S) = \emptyset$, and the condition that if one of A_2, A_3 equals τ, then the other must be ϵ. Since $a \notin \mathtt{fn}(T)$, then by Lemma A.2 $a \notin \mathtt{fn}(\mathbf{K}_3)$. Now, since $a \notin \mathtt{fn}(T)$, by Lemma A.4, we have that $a \in \mathtt{fn}(A_3)$ only if $a \in \mathtt{fn}(\beta)$ where β occurs in A_3. But then we cannot have $a\mathtt{fon}(A_3)$, for $\mathtt{fon}(\beta) = \emptyset$, by definition. Hence we have $a \notin \mathtt{fon}(A_3)$, which implies, since $a \in \mathtt{fon}(A_1) = \mathtt{fon}(A_2) \cup \mathtt{fon}(A_3)$, that $a \in \mathtt{fon}(A_2)$. We can thus apply L.NU.F to get $\nu a.S \xrightarrow{\nu a.\mathbf{K}_2 : \nu a.A_2} S'$. Now, if necessary by α-converting T, we can always take $a \notin \mathtt{bn}(T)$, which means by Lemma A.2 that $a \notin \mathtt{bn}(A_3)$. Thus, we have $\mathtt{bn}(\nu a.A_2) \cap \mathtt{bn}(A_3) = \emptyset$. Also $\mathtt{bn}(\overline{\nu}a.A_2) \cap \mathtt{fn}(T)$ for $a \notin \mathtt{fn}(T)$, and $\mathtt{bn}(A_3) \cap \mathtt{fn}(\overline{\nu}a.S)$. Hence we can apply L.PAR to get:

    $$(\nu a.S) \mid T \xrightarrow{(\nu a.\mathbf{K}_2) \mid \mathbf{K}_3 : (\nu a.A_2) \mid A_3} S' \mid T'$$

    Now, since $a \notin \mathtt{fon}(A_3)$, and since $\mathtt{bn}(\nu a.A_2) \cap \mathtt{bn}(A_3) = \emptyset$, we have by the definition of action equivalence $(\overline{\nu}a.A_2 \mid A_3 \equiv \overline{\nu}a.A_2 \mid A_3$. Hence we have found $Q' = S' \mid T' = P'$, $\mathbf{K}' = (\overline{\nu}a.\mathbf{K}_2) \mid \mathbf{K}_3 \equiv \overline{\nu}a.\mathbf{K}_2 \mid \mathbf{K}_3 = \overline{\nu}a.\mathbf{K}_1 = \mathbf{K}$, and $B = (\overline{\nu}a.A_2) \mid A_3 \equiv \overline{\nu}a.A_2 \mid A_3 = \overline{\nu}a.A_1 = A$, as required.
  - In case (ii), we should have $A_1 = \tau$, but since $a \in \mathtt{fon}(A_1)$, we have a contradiction, hence this case is void.
  - In case (iii), we reason as in case (ii) above.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.LOC. In this case, we have $P = a \bullet R$, $R \xrightarrow{\mathbf{K}_1 : A_1} R'$, $\mathbf{K} = a \bullet \mathbf{K}_1$, $A = \langle A_1 \rangle$, $P' = a \bullet R'$. Now, $P = a \bullet R \equiv Q$ could only have been derived through S.NU.KELL, hence $R = \nu b.S$, $Q = \nu b.a \bullet S$, and $b \neq a$. But then $R \xrightarrow{\mathbf{K}_1 : A_1} R'$ could only have been derived via (i) L.NU.NF, (ii) L.NU.F, (iii) L.RED.S, (iv) L.RED.H, (v) L.NULL.

- In case (i), we have $S \xrightarrow{\mathbf{K}_2 : A_1} S'$, $\mathbf{K}_1 = \overline{\nu}b.\mathbf{K}_2$, $R' = \nu b.S'$, $b \notin \mathtt{fn}(A_1)$. Applying rule L.LOC we get: $a \bullet S \xrightarrow{a \bullet \mathbf{K}_2 : \langle A_1 \rangle} a \bullet S'$. Applying rule L.NU.NF, we get: $\nu b.a \bullet S \xrightarrow{\overline{\nu}b.a \bullet \mathbf{K}_2 : \langle A_1 \rangle} \nu b.a \bullet S'$. Hence we have found $Q' = \nu b.a \bullet S' \equiv a \bullet \nu b.S' = a \bullet R' = P'$, $\mathbf{K}' = \overline{\nu}b.a \bullet \mathbf{K}_2 \equiv a \bullet \overline{\nu}b.\mathbf{K}_2 = a \bullet \mathbf{K}_1 = \mathbf{K}$, $B = \langle A_1 \rangle = A$, as required.
- In case (ii), we have $S \xrightarrow{\mathbf{K}_2 : A_2} R'$, $\mathbf{K}_1 = \overline{\nu}b.\mathbf{K}_2$, $A_1 = \overline{\nu}b.A_2$, $b \in \mathtt{fn}(A_2)$. Applying rule L.LOC we get: $a \bullet S \xrightarrow{a \bullet \mathbf{K}_2 : \langle A_2 \rangle} a \bullet R'$. Applying rule L.NU.F, we get: $\nu b.a \bullet S \xrightarrow{\overline{\nu}b.a \bullet \mathbf{K}_2 : \langle \overline{\nu}b.A_2 \rangle} a \bullet R'$. Hence we have found $Q' = a \bullet R' = P'$, $\mathbf{K}' = \overline{\nu}b.a \bullet \mathbf{K}_2 \equiv a \bullet \overline{\nu}b.\mathbf{K}_2 = a \bullet \mathbf{K}_1 = \mathbf{K}$, $B = \langle \overline{\nu}b.A_2 \rangle = \langle A_1 \rangle = A$, as required.
- In case (iii), we have $\mathbf{K}_1 = R$, $A_1 = \tau$, $R \xrightarrow{\mathbf{K}_2 : A_2} R''$, $\widetilde{c} = \mathtt{bn}(A_2)$, $R' = \nu\widetilde{c}.R''$, $\lceil A_2 \rceil = A'_1 \mid (\xi, \mathbf{C}) \mid A'_2$, $\mathbf{C} = \lceil \mathbf{K}_2 \rceil$, $\xi\theta_{\mathbf{C}} = [A'_1 \mid A'_2]$, $|A'_1| \leq 1$ and $|A'_2| \leq 1$. Now $R \xrightarrow{\mathbf{K}_2 : A_2} R''$ could only have been derived through (1) L.NU.NF, or (2) L.NU.F. Note that we can ensure $\widetilde{c} \cap \{a\} = \emptyset$ by a suitable α-conversion of R.
  - In case (1), we have $S \xrightarrow{\mathbf{K}_3 : A_3} S'$, $\mathbf{K}_2 = \overline{\nu}b.\mathbf{K}_3$, $A_2 = A_3$, $R'' = \nu b.S'$, $b \notin \mathtt{fn}(A_3)$. We have $\lceil A_3 \rceil = \lceil A_2 \rceil = A'_1 \mid (\xi, \theta_{\mathbf{C}}) \mid A'_2$, $\mathbf{C} = \lceil \mathbf{K}_2 \rceil = \lceil \overline{\nu}b.\mathbf{K}_3 \rceil = \lceil \mathbf{K}_3 \rceil$, hence we can apply L.RED.S to get: $S \xrightarrow{S : \tau} \nu\widetilde{c}.S'$. Now by L.LOC: $a \bullet S \xrightarrow{a \bullet S : \tau} a \bullet \nu\widetilde{c}.S'$. And by L.NU.BF:
    $$\nu b.a \bullet S \xrightarrow{\overline{\nu}b.a \bullet S : \tau} \nu b.a \bullet \nu\widetilde{c}.S'$$
    Note that in this case $b \notin \mathtt{fn}(A_3)$, hence by Lemma A.2, it it possible to ensure that $b \notin \widetilde{c}$ by a suitable α-conversion of S. Hence we have found $Q' = \nu b.a \bullet \nu\widetilde{c}.S' \equiv a \bullet \nu\widetilde{c}.\nu b.S' = a \bullet R' = P'$, $\mathbf{K} = \overline{\nu}b.a \bullet S \equiv a \bullet \overline{\nu}b.S \equiv a \bullet \nu b.S = a \bullet R = a \bullet \mathbf{K}_1 = \mathbf{K}$, and $B = \tau = A$, as required.
  - In case (2), we have $S \xrightarrow{\mathbf{K}_3 : A_3} S'$, $\mathbf{K}_2 = \overline{\nu}b.\mathbf{K}_3$, $A_2 = \overline{\nu}b.A_3$, $R'' = S'$, $b \in \mathtt{fon}(A_3)$. We have $\lceil \overline{\nu}b.A_3 \rceil = \lceil A_2 \rceil = A'_1 \mid (\xi, \theta_{\mathbf{C}}) \mid A'_2$, $\mathbf{C} = \lceil \mathbf{K}_2 \rceil = \lceil \overline{\nu}b.\mathbf{K}_3 \rceil = \lceil \mathbf{K}_3 \rceil$, hence we can apply L.RED.S to get: $S \xrightarrow{S : \tau} \nu\widetilde{c}.S'$, with $\widetilde{c} = b\widetilde{c}'$. Now by L.LOC: $a \bullet S \xrightarrow{a \bullet S : \tau} a \bullet \nu\widetilde{c}'.S'$. And by L.NU.BF:
    $$\nu b.a \bullet S \xrightarrow{\overline{\nu}b.a \bullet S : \tau} \nu b.a \bullet \nu\widetilde{c}'.S'$$
    Hence we have found $Q' = \nu b.a \bullet \nu\widetilde{c}'.S' \equiv a \bullet \nu b.\nu\widetilde{c}'.S' = a \bullet \nu\widetilde{c}.R'' = a \bullet R' = P'$, $\mathbf{K} = \overline{\nu}b.a \bullet S \equiv a \bullet \overline{\nu}b.S \equiv a \bullet \nu b.S = a \bullet R = a \bullet \mathbf{K}_1 = \mathbf{K}$, and $B = \tau = A$, as required.
- In case (iv), we reason exactly as in case (iii) above.
- In case (v), we have $\mathbf{K}_1 = R$, $A = \epsilon$, $R' = R$, hence $P' = a \bullet R = P$, and $\mathbf{K} = a \bullet R \equiv P$. By L.NULL, we have $Q \xrightarrow{Q : \epsilon} Q$, hence we have found $Q' = Q \equiv P = P'$, $\mathbf{K}' = Q \equiv P \equiv \mathbf{K}$, and $B = \epsilon = A$, as required.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.PAR. In this case, we have $P = R_1 \mid R_2$, $R_1 \xrightarrow{\mathbf{K}_1 : A_1} R'_1$, $R_2 \xrightarrow{\mathbf{K}_2 : A_2} R'_2$, $P' = R'_1 \mid R'_2$, $\mathbf{K} = \mathbf{K}_1 \mid \mathbf{K}_2$, $A = A_1 \mid A_2$, $\mathtt{bn}(A_1) \cap \mathtt{bn}(A_2) = \emptyset$, $\mathtt{bn}(A_1) \cap \mathtt{fn}(P_2) = \emptyset$, $\mathtt{bn}(A_2) \cap \mathtt{fn}(P_1) = \emptyset$, and the condition that if one of A_2, A_3 equals τ, then the other must be ϵ. Now $P \equiv Q$ could only have been derived through (a) S.PAR.COMM, (b) S.PAR.ASSOC, (c) S.NU.PAR.

- In case (a), we have $Q = R_2 \mid R_1$. By rule L.PAR, we have:

$$R_2 \mid R_1 \xrightarrow{\mathbf{K}_2 \mid \mathbf{K}_1 : A_2 \mid A_1} R_2' \mid R_1'$$

  Hence we have found $Q' = R_2' \mid R_1' \equiv R_1' \mid R_2' = P'$, $\mathbf{K}' = \mathbf{K}_2 \mid \mathbf{K}_1 \equiv \mathbf{K}_1 \mid \mathbf{K}_2 = \mathbf{K}$, $B = A_2 \mid A_1 \equiv A_1 \mid A_2 = A$, as required.

- In case (b), we have $P = (S_1 \mid S_2) \mid R_2$, $R_1 = S_1 \mid S_2$ and $Q = S_1 \mid (S_2 \mid R_2)$ (or a symmetric case with R_2). Now, we must have had $R_1 \xrightarrow{\mathbf{K}_1 : A_1} R_1'$ through rule L.PAR, hence we have $S_1 \xrightarrow{\mathbf{K}_3 : A_3} S_1'$, $S_2 \xrightarrow{\mathbf{K}_4 : A_4} S_2'$, $R_1' = S_1' \mid S_2'$, $\mathbf{K}_1 = \mathbf{K}_3 \mid \mathbf{K}_4$, $A_1 = A_3 \mid A_4$, $\mathsf{bn}(A_3) \cap \mathsf{bn}(A_4) = \emptyset$, $\mathsf{bn}(A_3) \cap \mathsf{fn}(P_4) = \emptyset$, $\mathsf{bn}(A_4) \cap \mathsf{fn}(P_3) = \emptyset$. We can apply rule L.PAR to S_2 and R_2 to get:

$$S_2 \mid R_2 \xrightarrow{\mathbf{K}_4 \mid \mathbf{K}_2 : A_4 \mid A_2} S_2' \mid R_2'$$

  Applying L.PAR another time we get:

$$S_1 \mid (S_2 \mid R_2) \xrightarrow{\mathbf{K}_3 \mid (\mathbf{K}_4 \mid \mathbf{K}_2) : A_3 \mid (A_4 \mid A_2)} S_1' \mid (S_2' \mid R_2')$$

  Hence we have found $Q' = S_1' \mid (S_2' \mid R_2') \equiv (S_1' \mid S_2') \mid R_2' = R_1' \mid R_2' = P'$, $\mathbf{K}' = \mathbf{K}_3 \mid (\mathbf{K}_4 \mid \mathbf{K}_2) \equiv (\mathbf{K}_3 \mid \mathbf{K}_4) \mid \mathbf{K}_2 = \mathbf{K}_1 \mid \mathbf{K}_2 = \mathbf{K}$, $B = A_3 \mid (A_4 \mid A_2) \equiv (A_3 \mid A_4) \mid A_2 = A_1 \mid A_2 = A$, as required.

- In case (c), we have $R_1 = \nu a.S_1$, $a \notin \mathsf{fn}(R_2)$, and $Q = \nu a.S_1 \mid R_2$. Now, $R_1 \xrightarrow{\mathbf{K}_1 : A_1} R_1'$ could only have been derived through (i) L.NU.NF, (ii) L.NU.F (iii) L.RED.S, (iv) L.RED.H, (v) hnameL.Par.

  - In case (i), we have $S_1 \xrightarrow{\mathbf{K}_1' : A_1'} S_1'$, $\mathbf{K}_1 = \nu a.\mathbf{K}_1'$, $A_1 = A_1'$, $a \notin \mathsf{fn}(A_1')$, $R_1' = \nu a.S_1'$. We have $\mathsf{bn}(A_1') \cap \mathsf{bn}(A_2) = \emptyset$, $\mathsf{bn}(A_1') \cap \mathsf{fn}(R_2) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{fn}(S_1) = \emptyset$, for since $a \notin \mathsf{fn}(R_2)$ it is always possible to α-convert R_2 to ensure $a \notin \mathsf{bn}(R_2)$, which ensures by Lemma A.2 that $a \notin \mathsf{bn}(A_2)$. Also, we can always α-convert R_1 to ensure $a \notin \mathsf{fn}(A_2)$. Hence by L.PAR we get:

$$S_1 \mid R_2 \xrightarrow{\mathbf{K}_1' \mid \mathbf{K}_2 : A_1 \mid A_2} S_1' \mid R_2'$$

    By L.NU.NF we get

$$\nu a.S_1 \mid R_2 \xrightarrow{\nu a.\mathbf{K}_1' \mid \mathbf{K}_2 : A_1 \mid A_2} \nu a.S_1' \mid R_2'$$

    Since $a \notin \mathsf{fn}(R_2)$, then by Lemma A.2, $a \notin \mathsf{fn}(\mathbf{K}_2)$. Hence we have found $Q' = \nu a.S_1' \mid R_2' \equiv (\nu a.S_1') \mid R_2' = R_1' \mid R_2' = P'$, $\mathbf{K}' = \overline{\nu} a.\mathbf{K}_1' \mid \mathbf{K}_2 \equiv (\overline{\nu} a.\mathbf{K}_1') \mid \mathbf{K}_2 = \mathbf{K}_1 \mid \mathbf{K}_2 = \mathbf{K}$, $B = A_1 \mid A_2 = A$, as required.

  - In case (ii), we have $S_1 \xrightarrow{\mathbf{K}_1' : A_1'} S_1'$, $\mathbf{K}_1 = \nu a.\mathbf{K}_1'$, $A_1 = \overline{\nu} a.A_1'$, $a \in \mathsf{fon}(A_1')$, $R_1' = S_1'$. Now, we have $a \notin \mathsf{fn}(R_2)$ and, as in the subcase (1) above, we can ensure that a does not occur in R_2 and have $a \notin \mathsf{fn}(A_2)$. Thus we have $\mathsf{bn}(A_1') \cap \mathsf{bn}(A_2) = \emptyset$, $\mathsf{bn}(A_1') \cap \mathsf{fn}(R_2) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{fn}(S_1) = \emptyset$. Hence by L.PAR we get:

$$S_1 \mid R_2 \xrightarrow{\mathbf{K}_1' \mid \mathbf{K}_2 : A_1' \mid A_2} R_1' \mid R_2'$$

Since $a \in \mathsf{fon}(A'_1)$ hence we can apply L.NU.F to get:

$$\nu a.S_1 \mid R_2 \xrightarrow{\overline{\nu}a.\mathbf{K}'_1 \mid \mathbf{K}_2 : \overline{\nu}a.A'_1 \mid A_2} R'_1 \mid R'_2$$

Hence we have found $Q' = R'_1 \mid R'_2 = P', \mathbf{K}' = \overline{\nu}a.\mathbf{K}'_1 \mid \mathbf{K}_2 \equiv (\overline{\nu}a.\mathbf{K}'_1) \mid \mathbf{K}_2 = \mathbf{K}_1 \mid \mathbf{K}_2 = \mathbf{K}$, $B = \overline{\nu}a.A'_1 \mid A_2 \equiv (\overline{\nu}a.A'_1) \mid A_2 = A_1 \mid A_2 = A$, as required.

– In case (iii), we have $A_1 = \tau$, $\mathbf{K}_1 = R_1$. By the conditions in rule L.PAR, we then have $A_2 = \epsilon$ and $R'_2 = R_2$. Also, we have $R_1 \xrightarrow{\mathbf{K}'_1 : A'_1} R''_1$, $R'_1 = \nu\widetilde{c}.R''_1$, $\widetilde{c} = \mathsf{bn}(A'_1)$, $\lceil A'_1 \rceil = B'_1 \mid (\xi, \theta_{\mathbf{C}}) \mid B'_2$, $\mathbf{C} = \lceil \mathbf{K}'_1 \rceil$, $\xi\theta_{\mathbf{C}} \equiv [B'_1 \mid B'_2]$. Now, since $R_1 = \nu a.S_1$, $R_1 \xrightarrow{\mathbf{K}'_1 : A'_1} R''_1$ could only have been derived via (1) L.NU.NF or (2) L.NU.F.

In case (1), we have $S_1 \xrightarrow{\mathbf{K}''_1 : A''_1} S'_1$, with $a \notin \mathsf{fn}(A''_1$, $R''_1 = \nu a.S'_1$, $A''_1 = A'_1$, $\mathbf{K}'_1 = \overline{\nu}a.\mathbf{K}''_1$. Note that $\lceil A''_1 \rceil = \lceil A'_1 \rceil = B'_1 \mid (\xi, \theta_{\mathbf{C}}) \mid B'_2$. Hence, because of L.TRIG, we have $S_1 \xrightarrow{\mathbf{K}''_1 : A'''_1} S'_1$ with A'''_1 identical to A''_1 except that the action $(\xi, \theta_{\mathbf{C}})$ occurring in A''_1 (which is unique by Lemma A.3) has been replaced by $(\xi, \theta_{\mathbf{C}'})$ and $\mathbf{C}' = \lceil \mathbf{K}''_1 \mid R_2 \rceil$. By the definition of action equivalence, we have $A'''_1 \equiv A'_1$. Now, since $A_2 = \epsilon$, we have $\mathsf{bn}(A''_1) \cap \mathsf{bn}(A_2) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{fn}(S_1) = \emptyset$. Also, it always possible to α-convert S_1 to ensure $\mathsf{bn}(A''_1) \cap \mathsf{fn}(R_2) = \emptyset$. Hence we can apply rule L.PAR to get:

$$S_1 \mid R_2 \xrightarrow{\mathbf{K}''_1 \mid \mathbf{K}_2 : A'''_1} S'_1 \mid R_2$$

Now by L.NU.NF we get:

$$\nu a.S_1 \mid R_2 \xrightarrow{\overline{\nu}a.\mathbf{K}''_1 \mid \mathbf{K}_2 : A'''_1} \nu a.S'_1 \mid R_2$$

Now, $\lceil \overline{\nu}a.\mathbf{K}''_1 \mid R_2 \rceil = \lceil \mathbf{K}''_1 \mid R_2 \rceil = \mathbf{C}'$ and, since for all $\mathbf{C}$, $\mathbf{E}$, $\theta_{\mathbf{E}[\mathbf{C}]} = \theta_{\mathbf{C}}$, we have $\xi\theta_{\mathbf{C}'} = \xi\theta_{\mathbf{C}}$. We can the apply L.RED.S to get:

$$\nu a.S_1 \mid R_2 \xrightarrow{\nu a.S_1 \mid R_2 : \tau} \nu\widetilde{c}.\nu a.S'_1 \mid R_2$$

Since we have ensured $\widetilde{c} = \mathsf{bn}(A'_1)$ is such that $\widetilde{c} \cap \mathsf{fn}(R_2) = \emptyset$, then we have found $Q' = \nu\widetilde{c}.\nu a.S'_1 \mid R_2 \equiv (\nu\widetilde{c}.\nu a.S'_1) \mid R_2 = (\nu\widetilde{c}.R''_1) \mid R_2 = R'_1 \mid R_2 = P'$, $\mathbf{K}' = \nu a.S_1 \mid R_2 \equiv (\nu a.S_1) \mid R_2 = \mathbf{K}$, and $B = \tau = A$, as required.

In case (2), we have $S_1 \xrightarrow{\mathbf{K}''_1 : A''_1} S'_1$, with $a \in \mathsf{fon}(A''_1$, $R''_1 = S'_1$, $A'_1 = \overline{\nu}a.A''_1$, $\mathbf{K}'_1 = \overline{\nu}a.\mathbf{K}''_1$. Note that $\lceil A''_1 \rceil = \lceil A'_1 \rceil = B'_1 \mid (\xi, \theta_{\mathbf{C}}) \mid B'_2$. Hence, because of L.TRIG, we have $S_1 \xrightarrow{\mathbf{K}''_1 : A'''_1} S'_1$ with A'''_1 identical to A''_1 except that the action $(\xi, \theta_{\mathbf{C}})$ occurring in A''_1 (which is unique by Lemma A.3) has been replaced by $(\xi, \theta_{\mathbf{C}'})$ and $\mathbf{C}' = \lceil \mathbf{K}''_1 \mid R_2 \rceil$. By the definition of action equivalence, we have $A'''_1 \equiv A'_1$. Now, since $A_2 = \epsilon$, we have $\mathsf{bn}(A''_1) \cap \mathsf{bn}(A_2) = \emptyset$, $\mathsf{bn}(A_2) \cap \mathsf{fn}(S_1) = \emptyset$. Also, it always possible to α-convert S_1 to ensure $\mathsf{bn}(A''_1) \cap \mathsf{fn}(R_2) = \emptyset$. Hence we can apply rule L.PAR to get:

$$S_1 \mid R_2 \xrightarrow{\mathbf{K}''_1 \mid \mathbf{K}_2 : A'''_1} S'_1 \mid R_2$$

Now by L.NU.F we get:

$$\nu a.S_1 \mid R_2 \xrightarrow{\overline{\nu}a.\mathbf{K}_1'' \mid \mathbf{K}_2 : \overline{\nu}a.A_1'''} S_1' \mid R_2$$

Now, $\lceil \overline{\nu}a.\mathbf{K}_1'' \mid R_2 \rceil = \lceil \mathbf{K}_1'' \mid R_2 \rceil = \mathbf{C}'$ and, since for all $\mathbf{C}$, $\mathbf{E}$, $\theta_{\mathbf{E}[\mathbf{C}]} = \theta_{\mathbf{C}}$, we have $\xi\theta_{\mathbf{C}'} = \xi\theta_{\mathbf{C}}$. We can the apply L.RED.S to get:

$$\nu a.S_1 \mid R_2 \xrightarrow{\nu a.S_1 \mid R_2 : \tau} \nu\widetilde{c}.S_1' \mid R_2$$

Since we have ensured $\widetilde{c} = a\widetilde{c}' = \mathsf{bn}(A_1')$ is such that $\widetilde{c} \cap \mathsf{fn}(R_2) = \emptyset$, then we have found $Q' = \nu\widetilde{c}.S_1' \mid R_2 \equiv (\nu\widetilde{c}.S_1') \mid R_2 = (\nu\widetilde{c}.R_1'') \mid R_2 = R_1' \mid R_2 = P'$, $\mathbf{K}' = \nu a.S_1 \mid R_2 \equiv (\nu a.S_1) \mid R_2 = \mathbf{K}$, and $B = \tau = A$, as required.

- In case (iv), we reason as in case (iii) above.
- In case (v), we have $\mathbf{K}_1 = R_1$, $A_1 = \epsilon$, $R_1' = R_1$, hence $P' = R_1 \mid R_2'$, and $\mathbf{K} = R_1 \mid \mathbf{K}_2$, $A = A_2$. By L.NULL, we have $S_1 \xrightarrow{S_1 : \epsilon} S_1$. Hence by L.PAR

$$S_1 \mid R_2 \xrightarrow{S_1 \mid \mathbf{K}_2 : A_2} S_1 \mid R_2'$$

Since $a \notin fn(()R_2)$, it is always possible to α-convert R_1 so that $a \notin \mathsf{fn}(A_2)$ and thus $a \notin \mathsf{fn}(R_2')$, we can apply L.NU.NF to get:

$$\nu a.S_1 \mid R_2 \xrightarrow{\overline{\nu}a.S_1 \mid \mathbf{K}_2 : A_2} \nu a.S_1 \mid R_2'$$

Hence we have found $Q' = \nu a.S_1 \mid R_2' \equiv (\nu a.S_1) \mid R_2' = P'$, $\mathbf{K}' = \overline{\nu}a.S_1 \mid \mathbf{K}_2 \equiv (\overline{\nu}a.S_1) \mid \mathbf{K}_2 \equiv (\nu a.S_1) \mid \mathbf{K}_2 = R_1 \mid \mathbf{K}_2 = \mathbf{K}$, and $B = A_2 = A$, as required.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.RED.S. In this case, we have $A = \tau$, $\mathbf{K} = P$, $P \xrightarrow{\mathbf{K}_1 : A'} P''$ with $P' = \nu\widetilde{c}.P''$, $\widetilde{c} = \mathsf{bn}(A')$, $\lceil A' \rceil = A_1 \mid (\xi, \theta_{\mathbf{C}}) \mid A_2$, $|A_1| \leq 1$, $|A_2| \leq 1$, $\mathbf{C} = \lceil \mathbf{K}_1 \rceil$, $\xi\theta_{\mathbf{C}} = [A_1 \mid A_2]$. Now, $P \equiv Q$ could only have been derived by one of the following rules (a) S.NIL, (b) S.NU.KELL, (c) S.NU.COMM, (d) S.NU.PAR.

- In case (a), we have $P = \nu a.\mathbf{0}$ and we have a contradiction for the only possible transition for P involves L.NULL and we have $A \neq \epsilon$. Hence this case is void.
- In case (b), we have $P = b \bullet \nu a.R$, $Q = \nu a.b \bullet R$, $b \neq a$ (or the symmetric case with $P = \nu a.b \bullet R$ and $Q = b \bullet \nu a.R$, which is handled similarly). But then $P \xrightarrow{\mathbf{K}_1 : A'} P''$ could only have been derived through (i) L.ACT or (ii) L.LOC. Case (i) is void for it would imply $A' = b \bullet \nu a.R$, which is not consistent with the fact that $\lceil A' \rceil = A_1 \mid A_0 \mid A_2$. Case (ii) is similarly void for it would imply $A' = \langle A'' \rangle$, which is also not consistent with the fact that $\lceil A' \rceil = A_1 \mid A_0 \mid A_2$. Hence the whole case is void.
- In case (c), we have $P = \nu a.\nu b.R$, $Q = \nu b.\nu a.R$, $b \neq a$. Now, $P \xrightarrow{\mathbf{K}_1 : A'} P''$ could only have been derived using two successive applications of rules L.NU.NF and L.NU.F. We thus have four cases to consider (i) L.NU.NF twice, (ii) L.NU.F twice, (iii) L.NU.NF followed by L.NU.F, (iv) L.NU.F followed by L.NU.NF. The four cases are handled similarly. We just consider the first two.

– In case (i), we have $R \xrightarrow{\mathbf{K}_2 : A'} R'$, with $\mathbf{K}_1 = \overline{\nu}a.\overline{\nu}b.\mathbf{K}_2$, $P'' = \nu a.\nu b.R'$. We can apply L.NU.NF twice to get $\nu b.\nu a.R \xrightarrow{\overline{\nu}b.\overline{\nu}a.\mathbf{K}_2 : A'} \nu b.\nu a.R'$. Since $\lceil \overline{\nu}b.\overline{\nu}a.\mathbf{K}_2 \rceil = \lceil \mathbf{K}_2 \rceil = \lceil \mathbf{K}_1 \rceil = \mathbf{C}$, we can apply L.RED.S to get:

$$\nu b.\nu a.R \xrightarrow{\nu b.\nu a.R : \tau} \nu \widetilde{c}.\nu b.\nu a.R'$$

Hence we have found $Q' = \nu \widetilde{c}.\nu b.\nu a.R' \equiv \nu \widetilde{c}.\nu a.\nu b.R' = \nu \widetilde{c}.P'' = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, $B = \tau = A$, as required.

– In case (ii), we have $R \xrightarrow{\mathbf{K}_2 : A''} P''$, with $\mathbf{K}_1 = \overline{\nu}a.\overline{\nu}b.\mathbf{K}_2$, $A' = \overline{\nu}a.\overline{\nu}b.A''$, . We can apply L.NU.F twice to get $\nu b.\nu a.R \xrightarrow{\nu b.\nu a.\mathbf{K}_2 : \nu b.\nu a.A''} P''$. We have $\lceil \nu b.\nu a.A'' \rceil = \lceil \nu a.\nu b.A'' \rceil = \lceil A' \rceil$, and $\lceil \overline{\nu}b.\overline{\nu}a.\mathbf{K}_2 \rceil = \lceil \mathbf{K}_2 \rceil = \lceil \mathbf{K}_1 \rceil = \mathbf{C}$, hence by L.RED.S we get:

$$\nu b.\nu a.R \xrightarrow{\nu b.\nu a.R : \tau} \nu \widetilde{c}.P''$$

Hence we have found $Q' = \nu \widetilde{c}.P'' = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, $B = \tau = A$, as required.

- In case (d), we have $P = (\nu a.R) \mid S$, $Q = \nu a.R \mid S$, $a \notin \mathtt{fn}(S)$ (or the symmetric case $P = \nu a.R \mid S$, $Q = (\nu a.R) \mid S$, which is handled similarly). Now, $P \xrightarrow{\mathbf{K}_1 : A'} P''$ could only have been derived using the following rules: (i) L.NU.NF followed by L.PAR, (ii) L.NU.F followed by L.PAR.

– In case (i), we have $R \xrightarrow{\mathbf{K}_2 : A'_2} R'$, $S \xrightarrow{\mathbf{K}_3 : A'_3} S'$, $a \notin \mathtt{fn}(A'_2)$, $\mathbf{K}_1 = (\overline{\nu}a.\mathbf{K}_2) \mid \mathbf{K}_3$, $A' = A'_2 \mid A'_3$, $P'' = (\nu a.R') \mid S'$, $\mathtt{bn}(A'_2) \cap \mathtt{bn}(A'_3) = \emptyset$, $\mathtt{bn}(A'_2) \cap \mathtt{fn}(S) = \emptyset$, $\mathtt{bn}(A'_3) \cap \mathtt{fn}(\nu a.R) = \emptyset$. Now, we can always α-convert $\nu a.R$ and S to ensure $a \not\in \mathtt{fn}(A_3) \cup \mathtt{bn}(S)$. Hence we have $\mathtt{bn}(A'_3) \cap \mathtt{fn}(R) = \emptyset$, and we can apply L.PAR to get:

$$R \mid S \xrightarrow{\mathbf{K}_2 \mid \mathbf{K}_3 : A'_2 \mid A'_3} R' \mid S'$$

Now $a \notin \mathtt{fn}(A'_3)$, hence $a \notin \mathtt{fn}(A'_2 \mid A'_3)$ and we can apply L.NU.NF to get:

$$\nu a.R \mid S \xrightarrow{\overline{\nu}a.\mathbf{K}_2 \mid \mathbf{K}_3 : A'_2 \mid A'_3} \nu a.R' \mid S'$$

Since $A' = A'_2 \mid A'_3$, and $\lceil \overline{\nu}a.\mathbf{K}_2 \mid \mathbf{K}_3 \rceil = \lceil (\overline{\nu}a.\mathbf{K}_2) \mid \mathbf{K}_3 \rceil = \lceil \mathbf{K}_2 \mid \mathbf{K}_3 \rceil$, we can apply L.RED.S to get: $\nu a.R \mid S \xrightarrow{Q : \tau} \nu \widetilde{c}.\nu a.R' \mid S'$. Hence we have found $Q' = \nu \widetilde{c}.\nu a.R' \mid S' \equiv \nu \widetilde{c}.P'' = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, $B = \tau = A$, as required.

– In case (ii), we have $R \xrightarrow{\mathbf{K}_2 : A'_2} R'$, $S \xrightarrow{\mathbf{K}_3 : A'_3} S'$, $a \in \mathtt{fon}(A'_2)$, $\mathbf{K}_1 = (\nu a.\mathbf{K}_2) \mid \mathbf{K}_3$, $A' = (\nu a.A'_2) \mid A'_3$, $P'' = R' \mid S'$, $\mathtt{bn}(A'_2) \cap \mathtt{bn}(A'_3) = \emptyset$, $\mathtt{bn}(A'_2) \cap \mathtt{fn}(S \mid A'_3) = \emptyset$, $\mathtt{bn}(A'_3) \cap \mathtt{fn}(R \mid A'_2) = \emptyset$. As in the subcase (i) above, we can ensure $a \notin \mathtt{fn}(A_3) \cup \mathtt{bn}(S)$. Hence, we can apply L.PAR to get:

$$R \mid S \xrightarrow{\mathbf{K}_2 \mid \mathbf{K}_3 : A'_2 \mid A'_3} R' \mid S'$$

Now, $a \notin \mathtt{fn}(A'_3)$ but $a \in \mathtt{fon}(A'_2)$ and we can apply L.NU.F to get:

$$\nu a.R \mid S \xrightarrow{\overline{\nu}a.\mathbf{K}_2 \mid \mathbf{K}_3 : \overline{\nu}a.A'_2 \mid A'_3} R' \mid S'$$

Since $\lceil \overline{\nu}a.A'_2 \mid A'_3 \rceil = \lceil A'_2 \mid A'_3 \rceil = \lceil A' \rceil$ and $\lceil \overline{\nu}a.\mathbf{K}_2 \mid \mathbf{K}_3 \rceil = \lceil \mathbf{K}_2 \mid \mathbf{K}_3 \rceil = \lceil \mathbf{K}_1 \rceil$, we can apply L.RED.S to get: $\nu a.R \mid S \xrightarrow{Q : \tau} \nu\widetilde{c}.R' \mid S'$. Hence we have found $Q' = \nu\widetilde{c}.R' \mid S' = \nu\widetilde{c}.P'' = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, $B = \tau = A$, as required.

Case: $P \xrightarrow{\mathbf{K} : A} P'$ **derived by** L.RED.H. This case is handled similarly to the previous one (derivation by L.RED.S).

We can now proceed to the final steps of the proof by considering derivations of $P \equiv Q$ involving rule S.CONTEXT. We proceed by induction on the form of $\mathbf{K}$ contexts.

Case $P = \nu a.R$ **and** $Q = \nu a.S$**,** $R \equiv S$. In this case $P \xrightarrow{\mathbf{K} : A} P'$ could only have been derived through one of (i) L.NU.NF, (ii) L.NU.F, (iii) L.RED.S, (iv) L.RED.H, (v) L.NULL.

- In case (i), we have $R \xrightarrow{\mathbf{K}_1 : A} R'$, with $a \notin \mathtt{fn}(A)$, $P' = \nu a.R'$, $\mathbf{K} = \overline{\nu}a.\mathbf{K}_1$. By induction hypothesis, we then have $S \xrightarrow{\mathbf{K}'_1 : B} S'$ with $\mathbf{K}'_1 \equiv \mathbf{K}_1$, $B \equiv A$, $S \equiv R$ and $S' \equiv R'$. Applying L.NU.NF we get: $\nu a.S \xrightarrow{\overline{\nu}a.\mathbf{K}'_1 : B} \nu a.S'$. Hence we have found $Q' = \nu a.S' \equiv \nu a.R' = P'$, $\mathbf{K}' = \overline{\nu}a.\mathbf{K}'_1 \equiv \overline{\nu}a.\mathbf{K}_1 = \mathbf{K}$, $B \equiv A$, as required.
- In case (ii), we have $R \xrightarrow{\mathbf{K}_1 : A'} P'$, with $a \in \mathtt{fon}(A')$, $A = \overline{\nu}a.A'$, $\mathbf{K} = \overline{\nu}a.\mathbf{K}_1$. By induction hypothesis, we then have $S \xrightarrow{\mathbf{K}'_1 : B'} S'$ with $\mathbf{K}'_1 \equiv \mathbf{K}_1$, $B' \equiv A'$, and $S \equiv R$, $S' \equiv P'$. Applying L.NU.F we get: $\nu a.S \xrightarrow{\overline{\nu}a.\mathbf{K}'_1 : \overline{\nu}a.B'} S'$. Hence we have found $Q' = S' \equiv= P'$, $\mathbf{K}' = \overline{\nu}a.\mathbf{K}'_1 \equiv \overline{\nu}a.\mathbf{K}_1 = \mathbf{K}$, $B = \overline{\nu}a.B' \equiv \overline{\nu}a.A' = A$, as required.
- In case (iii), we have $\mathbf{K} = P$, $A = \tau$, $P \xrightarrow{\mathbf{K}_1 : A_1} P_1$, $P' = \nu\widetilde{c}.P_1$, $\widetilde{c} = \mathtt{bn}(A_1)$, $\lceil A_1 \rceil = B_1 \mid (\xi, \theta_{\mathbf{C}}) \mid B_2$, $\mathbf{C} = \lceil \mathbf{K}_1 \rceil$, $\xi\theta_{\mathbf{C}} = [B_1 \mid B_2]$. Now, $P \xrightarrow{\mathbf{K}_1 : A_1} P_1$ could only have been derived using one of (1) L.NU.NF or (2) L.NU.F.
  - In case (1), we have $R \xrightarrow{\mathbf{K}_2 : A_2} R'$, with $a \notin \mathtt{fn}(A_2)$, $A_1 = A_2$, $P_1 = \nu a.R'$, $\mathbf{K}_1 = \overline{\nu}a.\mathbf{K}_2$. By induction hypothesis, we then have $S \xrightarrow{\mathbf{K}'_2 : B'} S'$ with $\mathbf{K}'_2 \equiv \mathbf{K}_2$, $B' \equiv A_2 = A_1$, and $S' \equiv R'$. Since $B' \equiv A_1$, we have $\lceil B' \rceil \equiv \lceil A_1 \rceil$. Applying L.NU.NF we get: $\nu a.S \xrightarrow{\overline{\nu}a.\mathbf{K}'_2 : B'} \nu a.S'$. Since $\lceil B' \rceil \equiv B_1 \mid (\xi, \theta_{\mathbf{C}}) \mid B_2$, then some $(\zeta, \theta_{\mathbf{C}'})$ must occur in B', such that $\xi \equiv \zeta$ and $\mathbf{C}' \equiv \mathbf{C}$. Hence we can apply rule L.RED.S to get: $\nu a.S \xrightarrow{\nu a.S : \tau} \nu\widetilde{c}.\nu a.S'$ Hence we have found $Q' = \nu\widetilde{c}.\nu a.S' \equiv \nu\widetilde{c}.\nu a.R' = P'$, $\mathbf{K}' = \nu a.S \equiv \nu a.R = P\mathbf{K}$, $B = \tau = A$, as required.
  - In case (2), we have $R \xrightarrow{\mathbf{K}_2 : A_2} R'$, with $a \in \mathtt{fon}(A_2)$, $A_1 = \nu a.A_2$, $P_1 = R'$, $\mathbf{K}_1 = \nu a.\mathbf{K}_2$. By induction hypothesis, we then have $S \xrightarrow{\mathbf{K}'_2 : B'} S'$ with $\mathbf{K}'_2 \equiv \mathbf{K}_2$, $B' \equiv A_2$,

and $S' \equiv R'$. Applying L.NU.F we get: $\nu a.S \xrightarrow{\nu a.\mathbf{K}'_2 : \nu a.B'} S'$. Since $B' \equiv A_2$, we have $\lceil \nu a.B' \rceil \equiv \lceil \nu a.A_2 \rceil \lceil A_1 \rceil$, and some $(\zeta, \theta_{\mathbf{C}'})$ must occur in B', such that $\xi \equiv \zeta$ and $\mathbf{C}' \equiv \mathbf{C}$. Hence we can apply rule L.RED.S to get: $\nu a.S \xrightarrow{\nu a.S : \tau} \nu \widetilde{c}.S'$ Hence we have found $Q' = \nu \widetilde{c}.S' \equiv \nu \widetilde{c}.\nu a.R' = P'$, $\mathbf{K}' = \nu a.S \equiv \nu a.R = P\mathbf{K}$, $B = \tau = A$, as required.

- Case (iv) is handled similarly as case (iii) above.
- In case (v), we have $\mathbf{K} = P$, $A = \epsilon$, $P' = P$. By rule L.NULL, we have $Q \xrightarrow{Q : \epsilon} Q$, hence we have found $Q' = Q \equiv P = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, and $B = \epsilon = A$, as required.

Case $P = R \mid T$ **and** $Q = S \mid T$**,** $R \equiv S$. In this case $P \xrightarrow{\mathbf{K} : A} P'$ could only have been derived through one of (i) L.PAR, (ii) L.RED.S, (iii) L.RED.H, (iv) L.NULL.

- In case (i), we have $R \xrightarrow{\mathbf{K}_1 : A_1} R'$, $T \xrightarrow{\mathbf{K}_2 : A_2} T'$, $P' = R' \mid T'$, $A = A_1 \mid A_2$, $\mathbf{K} = \mathbf{K}_1 \mid \mathbf{K}_2$. By induction hypothesis, we have $S \xrightarrow{\mathbf{K}'_1 : B_1} S'$ with $S' \equiv R'$, $\mathbf{K}'_1 \equiv \mathbf{K}_1$ and $B_1 \equiv A_1$. We can now apply rule L.PAR to get: $S \mid T \xrightarrow{\mathbf{K}'_1 \mid \mathbf{K}_2 : B_1 \mid A_2} S' \mid T'$. Hence we have found $Q' = S' \mid T' \equiv R' \mid T' = P'$, $\mathbf{K}' = \mathbf{K}'_1 \mid \mathbf{K}_2 \equiv \mathbf{K}_1 \mid \mathbf{K}_2 = \mathbf{K}$, and $B = B_1 \mid A_2 \equiv A_1 \mid A_2 = A$, as required.
- In case (ii), we have $\mathbf{K} = P$, $A = \tau$, $P \xrightarrow{\mathbf{K}_1 : A_1} P''$, $P' = \nu \widetilde{c}.P''$, $\widetilde{c} = \mathsf{bn}(A_1)$, $\lceil A_1 \rceil = B_1 \mid (\xi, \theta_{\mathbf{C}}) \mid B_2$, $\mathbf{C} = \lceil \mathbf{K}_1 \rceil$, $\xi\theta_{\mathbf{C}} \equiv [B_1 \mid B_2]$. Now, $P \xrightarrow{\mathbf{K}_1 : A_1} P''$ could only have been obtained through rule L.PAR, hence we must have: $R \xrightarrow{\mathbf{K}_2 : A_2} R'$, $T \xrightarrow{\mathbf{K}_3 : A_3} T'$, $P'' = R' \mid T'$, $A_1 = A_2 \mid A_3$, $\mathbf{K}_1 = \mathbf{K}_2 \mid \mathbf{K}_3$, with additional conditions from the premise of rule L.PAR. By induction hypothesis, we must have $S \xrightarrow{\mathbf{K}'_2 : A'_2} S'$, with $S' \equiv R'$, $\mathbf{K}'_2 \equiv \mathbf{K}_2$, $A'_2 \equiv A_2$. We can apply L.PAR to get: $S \mid T \xrightarrow{\mathbf{K}'_2 \mid \mathbf{K}_3 : A'_2 \mid A_3} S' \mid T'$. Now the conditions of rule L.RED.S are still valid and applying L.RED.S we get: $S \mid T \xrightarrow{S \mid T : \tau} \nu \widetilde{c}.S' \mid T'$. Hence we have found $Q' = \nu \widetilde{c}.S' \mid T' \equiv \nu \widetilde{c}.R' \mid T' = P'$, $\mathbf{K}' = S \mid T = Q \equiv P = \mathbf{K}$, $B = \tau = A$, as required.
- Case (iii) is handled similarly as case (ii) above.
- In case (iv), we have $\mathbf{K} = P$, $A = \epsilon$, $P' = P$. By rule L.NULL, we have $Q \xrightarrow{Q : \epsilon} Q$, hence we have found $Q' = Q \equiv P = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, and $B = \epsilon = A$, as required.

Case $P = a \bullet R$ **and** $Q = a \bullet S$**,** $R \equiv S$. In this case $P \xrightarrow{\mathbf{K} : A} P'$ could only have been derived through one of (i) L.ACT, (ii) L.LOC, (iii) L.RED.H, (iv) L.NULL.

- In case (i), we have $\mathbf{K} = P = A = a \bullet R$, and $P' = \mathbf{0}$. But, by L.ACT, we have $Q \xrightarrow{Q : a \bullet S} \mathbf{0}$. Hence we have found $Q' = \mathbf{0} = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, and $B = a \bullet S \equiv a \bullet R = A$, as required.
- In case (ii), we have $R \xrightarrow{\mathbf{K}_1 : A_1} R'$, $\mathbf{K} = a \bullet \mathbf{K}_1$, $P' = a \bullet R'$, $A = \langle A_1 \rangle$. By induction hypothesis we must have: $S \xrightarrow{\mathbf{K}'_1 : B_1} S'$ with $S' \equiv R'$, $\mathbf{K}'_1 \equiv \mathbf{K}_1$, $B_1 \equiv A_1$. By L.LOC we get: $a \bullet S \xrightarrow{a \bullet \mathbf{K}'_1 : \langle B_1 \rangle} a \bullet S'$. Hence we have found $Q' = a \bullet S' \equiv a \bullet R' = P'$, $\mathbf{K}' = a \bullet \mathbf{K}'_1 \equiv a \bullet \mathbf{K}_1 = \mathbf{K}$, $B = \langle B_1 \rangle \equiv \langle A_1 \rangle = A$, as required.

- In case (iii), we have $\mathbf{K} = P$, $A = \tau$, $P \xrightarrow{\mathbf{K}_1 : A_1} P''$, $P' = \nu\widetilde{c}.P''$, $\widetilde{c} = \mathsf{bn}(A_1)$, $\lceil A_1 \rceil = B_3 \mid \langle B_1 \mid (\xi, \theta_{\mathbf{C}}) \mid B_2 \rangle \mid B_4$, $\mathbf{C} = \lceil \mathbf{K}_1 \rceil$, $\xi\theta_{\mathbf{C}} \equiv B_3 \mid [B_1 \mid B_2] \mid B_4$. Now, $P \xrightarrow{\mathbf{K}_1 : A_1} P''$ could only have been obtained through L.LOC, with $A_1 = \langle A_1' \rangle$, $R \xrightarrow{\mathbf{K}_2 : A_1'} R'$, $\mathbf{K}_1 = a \bullet \mathbf{K}_2$, $P'' = a \bullet R'$. By induction hypothesis, we must have $S \xrightarrow{\mathbf{K}_2' : B_1'} S'$, with $S' \equiv R'$, $\mathbf{K}_2' \equiv \mathbf{K}_2$, $B_1' \equiv A_1'$. By rule L.LOC we get: $a \bullet S \xrightarrow{a \bullet \mathbf{K}_2' : \langle B_1' \rangle} a \bullet S'$. Conditions of rule L.RED.H are still valid since $\lceil \langle B_1' \rangle \rceil \equiv \lceil A_1' \rceil = \lceil A_1 \rceil$ and $\lceil a \bullet \mathbf{K}_2' \rceil \equiv \lceil a \bullet \mathbf{K}_2 \rceil = \lceil \mathbf{K}_1 \rceil$. Applying L.RED.H, we get: $a \bullet S \xrightarrow{a \bullet S : \tau} \nu\widetilde{c}.a \bullet S'$. Hence we have found $Q' = \nu\widetilde{c}.a \bullet S' \equiv \nu\widetilde{c}.a \bullet R' = \nu\widetilde{c}.P'' = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, and $B = \tau = A$, as required.
- In case (iv), we have $\mathbf{K} = P$, $A = \epsilon$, $P' = P$. By rule L.NULL, we have $Q \xrightarrow{Q : \epsilon} Q$, hence we have found $Q' = Q \equiv P = P'$, $\mathbf{K}' = Q \equiv P = \mathbf{K}$, and $B = \epsilon = A$, as required.

Case $P = \xi \triangleright R$ **and** $Q = \xi \triangleright S$**,** $R \equiv S$. In this case $P \xrightarrow{\mathbf{K} : A} P'$ could only have been derived through L.TRIG. We thus have $\mathbf{K} = \cdot$, $A = (\xi, \theta_{\mathbf{K}_1})$ and $P' = R\theta_{\mathbf{K}_1}$. By L.TRIG, we have $Q \xrightarrow{\cdot : (\xi, \theta_{\mathbf{K}_1})} S\theta_{\mathbf{K}_1}$. By Lemma A.1, up to a possible renaming of bound names in R and S, $R \equiv S$ implies that we have $R\theta_{\mathbf{K}_1} \equiv S\theta_{\mathbf{K}_1}$, hence we have found $Q' \equiv P'$, $\mathbf{K}' = \cdot = \mathbf{K}$, and $B = (\xi, \theta_{\mathbf{K}_1}) = A$, as required.

This concludes the proof of Proposition 2.1.

□

Let $P \downarrow A$ mean $P \xrightarrow{\mathbf{K} : A} P'$ for some $\mathbf{K}, P'$. We abbreviate $\widetilde{A}$ a parallel composition of actions $\prod_{l \in L} A_l$, and $\widetilde{Q}$ a parallel composition $\prod_{l \in L} Q_l$ of Kell calculus processes. We abbreviate $\widetilde{Q} \downarrow \widetilde{A}$ the proposition $\bigwedge_{l \in L} Q_l \downarrow A_l$.

Lemma A.5 *Let $A \neq \epsilon, \tau$. Then we have*

1. *If $P \downarrow A$, with $\lceil A \rceil = \alpha$, then $P \equiv \nu\widetilde{c}.Q \mid R$ with $Q \downarrow \alpha$ and $R \downarrow \epsilon$.*
2. *If $P \downarrow A$, with $\lceil A \rceil = \beta$, then $P \equiv \nu\widetilde{c}.Q \mid R$ with $Q \downarrow \beta$ and $R \downarrow \epsilon$.*
3. *If $P \downarrow A$, with $\lceil A \rceil = \langle \alpha \rangle$, then $P \equiv \nu\widetilde{c}.a \bullet (Q \mid R) \mid S$ with $Q \downarrow \alpha$, $R \downarrow \epsilon$, $S \downarrow \epsilon$.*
4. *If $P \downarrow A$ with $\lceil A \rceil = \prod_{i \in I} \alpha_i$, then $P \equiv \nu\widetilde{c}.R \mid \prod_{i \in I} Q_i$ with $Q_i \downarrow \alpha_i$, $R \downarrow \epsilon$.*
5. *If $P \downarrow A$ with $\lceil A \rceil = \langle \prod_{i \in I} \alpha_i \rangle$, then $P \equiv \nu\widetilde{c}.R \mid a \bullet (R_i \mid \prod_{i \in I} Q_i)$ with $Q_i \downarrow \alpha_i$, $R \downarrow \epsilon$.*
6. *If $P \downarrow A$ with $\lceil A \rceil \equiv \beta \mid \prod_{i \in I} \alpha_i \mid \prod_{j \in J} \langle \widetilde{\alpha_j} \rangle$ then $P \equiv \nu\widetilde{d}.Q \mid S \mid \prod_{i \in I} Q_i \mid \prod_{j \in J} a_j \bullet (\widetilde{Q_j} \mid R_j)$ with $Q \downarrow \beta$, $S \downarrow \epsilon$, $Q_i \downarrow \alpha_i$, $\widetilde{Q_j} \downarrow \widetilde{\alpha_j}$, $R_j \downarrow \epsilon$.*
7. *If $P \downarrow A$ with $\lceil A \rceil \equiv \prod_{i \in I} \alpha_i \mid \langle \beta \mid \prod_{k \in K} \alpha_k \mid \prod_{j \in J} \langle \widetilde{\alpha_j} \rangle \rangle$ then $P \equiv \nu\widetilde{d}.S \mid \prod_{i \in I} Q_i \mid a \bullet (Q \mid T \mid \prod_{i \in I} Q_i \mid \prod_{j \in J} a_j \bullet (\widetilde{Q_j} \mid R_j))$ with $S \downarrow \epsilon$, $T \downarrow \epsilon$, $Q \downarrow \beta$, $S \downarrow \epsilon$, $Q_i \downarrow \alpha_i$, $\widetilde{Q_j} \downarrow \widetilde{\alpha_j}$, $Q_k \downarrow \alpha_k$ $R_j \downarrow \epsilon$.*

Proof: Properties 1 to 3 are proved by a simple induction on the derivation $P \xrightarrow{\mathbf{K} : A} P'$. Properties 4 and 5 are proved by a simple induction. Properties 6 and 7 are proved from properties 3, 4 and 5 by applying rule L.PAR. □

Lemma A.6 *If* $P \xrightarrow{\mathbf{K} : \tau} Q$, *then* $P \rightarrow Q$.

Proof: We proceed by induction on the depth of inference of $P \xrightarrow{\mathbf{K} : \tau} Q$. This transition could only have been inferred through one of the following rules: L.LOC, L.NU.NF, L.PAR, L.RED.S, L.RED.H.

- L.LOC. In this case we have $P = a \bullet R$, $Q = a \bullet S$, $\mathbf{K} = a \bullet \mathbf{K}'$, $R \xrightarrow{\mathbf{K}' : \tau} S$. By induction hypothesis, we have $R \rightarrow S$, hence by R.CONTEXT, $P \rightarrow Q$, as required.
- L.NU.NF. In this case, we have $P = \nu a.R$, $Q = \nu a.S$, $R \xrightarrow{\mathbf{K}' : \tau} S$, $\mathbf{K} = \overline{\nu} a.\mathbf{K}'$. By induction hypothesis, we have $R \rightarrow S$, hence by R.CONTEXT, $P \rightarrow Q$, as required.
- L.PAR. In this case we have $P = P_1 \mid P_2$, $Q = Q_1 \mid Q_2$, with $P_1 \xrightarrow{\mathbf{K}_1 : \tau} Q_1$, $P_2 \xrightarrow{\mathbf{K}_1 : \epsilon} P_2$ or $P_2 \xrightarrow{\mathbf{K}_1 : \tau} Q_2$, $P_1 \xrightarrow{\mathbf{K}_1 : \epsilon} P_1$. In the first case (the other is identical), we have by induction hypothesis $P_1 \rightarrow Q_1$, hence by R.CONTEXT we have $P = P_1 \mid P_2 \rightarrow Q_1 \mid P_2 = Q$, as required.
- L.RED.S. In this case we have $P \xrightarrow{\mathbf{K}' : A'} P' \lceil A' \rceil = A_1 \mid (xi, \theta_{\mathbf{C}}) \mid A_2$, $\mathbf{C} = \lceil \mathbf{K}' \rceil$, $\xi\theta_{\mathbf{C}} = [A_1 \mid A_2]$. Then by Lemma A.5(6), we have that $P \equiv \overline{P} = \nu \widetilde{d}.Q \mid S \mid \prod_{i \in I} Q_i \mid \prod_{j \in J} a_j \bullet (\widetilde{Q_j} \mid R_j)$ with $Q = \xi \triangleright Q'$, $Q \downarrow (\xi, \theta_{\mathbf{C}})$, $S \downarrow \epsilon$, $Q_i \downarrow \alpha_i$, $\widetilde{Q_j} \downarrow \widetilde{\alpha_j}$, $R_j \downarrow \epsilon$, $A_1 = \prod_{i \in I} \alpha_i$, $A_2 = \prod_{j \in J} \widetilde{\alpha_j}$. But then by L.RED.S, S.PAR.NIL and Theorem 1 we have that $P' \equiv \overline{P'} = \nu \widetilde{d}.Q'\theta_{\mathbf{C}} \mid S \mid \prod_{j \in J} a_j \bullet R_j$. Since $\overline{P} \rightarrow \overline{P'}$ by R.RED.S, we have by R.EQUIV that $P \rightarrow Q$, as required.
- L.RED.H. In this case, we reason as in the case L.RED.S above, using Lemma A.5(7) instead.

□

Lemma A.7 *If* $R \rightarrow S$, *then there exist* S', $\mathbf{K}$ *such that* $R \xrightarrow{\mathbf{K} : \tau} S'$ *and* $S' \equiv S$.

Proof: We proceed by induction on the depth of inference of $P \rightarrow Q$.

Case R.RED.S. In this case, it suffices to notice that rule L.RED.S applies to $R = \xi \triangleright P \mid N \mid Q$ with P, N, Q as in rule R.RED.S, and yields $R \xrightarrow{R : \tau} R'$, with $R' = P\theta_{\mathbf{C}} \mid Q'$, with $\theta_{\mathbf{C}}, Q'$ as in rule R.RED.S. Hence we have found $S' = R' = S$, as required.

Case R.EQUIV. In this case, we have $R \equiv P$, $P \rightarrow Q$, and $Q \equiv S$. By induction hypothesis we have $P \xrightarrow{\mathbf{K} : \tau} Q'$ and $Q' \equiv Q$. But by Theorem 1, since $R \equiv P$, there must exist R' such that $R \xrightarrow{\mathbf{K} : \tau} R'$ and $R' \equiv Q'$. Hence we have found $S' = R' \equiv Q' \equiv Q \equiv S$, as required.

Case R.CONTEXT. In this case, we assume $P \rightarrow Q$, $P = \mathbf{E}[R]$, $Q = \mathbf{E}[S]$, with $R \rightarrow S$, and we proceed by induction on the form of evaluation contexts.

- Case $P = a \bullet R$ and $Q = a \bullet S$, with $R \rightarrow S$. By induction hypothesis, we have $R \xrightarrow{\mathbf{K} : \tau} S'$ and $S' \equiv S$. But then by L.LOC we get: $a \bullet R \xrightarrow{\mathbf{K} : \tau} a actif S'$. Hence we have found $Q' = a \bullet S' \equiv a \bullet S = Q$, as required.
- Case $P = \nu a.R$ and $Q = \nu a.S$, with $R \rightarrow S$. By induction hypothesis, we have $R \xrightarrow{\mathbf{K} : \tau} S'$ and $S' \equiv S$. But then by L.NU.NF we get $\nu a.R \xrightarrow{\overline{\nu} a.\mathbf{K} : \tau} \nu a.S'$. Hence we have found $Q' = \nu a.S' \equiv \nu a.S = Q$, as required.

- Case $P = R \mid T$, and $Q = S \mid T$, with $R \to S$. By induction hypothesis, we have $R \xrightarrow{\mathbf{K} : \tau} S'$ and $S' \equiv S$. By L.NULL, we have $T \xrightarrow{T : \epsilon} T$ But then by L.PAR, we get $R \mid T \xrightarrow{\mathbf{K} \mid T : \tau} S' \mid T$. Hence we have found $Q' = S' \mid T \equiv S \mid T = Q$, as required.

□

Proof of Theorem 2: Theorem 1 results directy from the conjunction of Lemma A.6 and of Lemma A.7. □

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399