A Logical and Computational Theory of Located Resource

Matthew Collinson, Brian Monahan, David Pym

HP Laboratories
HPL-2008-74R1

Keyword(s):
Bunched logic, process algebra, resource semantics, location, systems modelling

Abstract:

Experience of practical systems modelling suggests that the key conceptual components of a model of a system are processes, resources, locations, and environment. In recent work, we have given a process-theoretic account of this view in which resources as well as processes are first-class citizens. This process calculus, SCRP, captures the structural aspects of the semantics of the Demos2k modelling tool. Demos2k represents environment stochastically using a wide range of probability distributions and queue-like data structures. Associated with SCRP is a (bunched) modal logic, MBI, which combines the usual additive connectives of Hennessy-Milner logic with their multiplicative counterparts. In this paper, we complete our conceptual framework by adding to SCRP and MBI an account of a notion of location that is simple, yet sufficiently expressive to capture naturally a wide range of forms of location, both spatial and logical. We also provide a description of an extension of the Demos2k tool to incorporate this notion of location.

External Posting Date: December 6, 2008 [Fulltext] Approved for External Publication
Internal Posting Date: December 6, 2008 [Fulltext]

Submitted to Journal of Logic and Computation

A Logical and Computational Theory of Located Resource

Matthew Collinson* Brian Monahan† David Pym‡§

November 27, 2008

Abstract

Experience of practical systems modelling suggests that the key conceptual components of a model of a system are processes, resources, locations, and environment. In recent work, we have given a process-theoretic account of this view in which resources as well as processes are first-class citizens. This process calculus, **SCRP**, captures the structural aspects of the semantics of the Demos2k modelling tool. Demos2k represents environment stochastically using a wide range of probability distributions and queue-like data structures. Associated with **SCRP** is a (bunched) modal logic, **MBI**, which combines the usual additive connectives of Hennessy-Milner logic with their multiplicative counterparts. In this paper, we complete our conceptual framework by adding to **SCRP** and **MBI** an account of a notion of location that is simple, yet sufficiently expressive to capture naturally a wide range of forms of location, both spatial and logical. We also provide a description of an extension of the Demos2k tool to incorporate this notion of location.

1 Introduction

In this paper, we are concerned with the theoretical foundations of systems modelling. We wish to focus on systems that feature agents that undergo state-change in a step-wise fashion (discrete event systems), and that make use of resource which is distributed around a space of locations. The techniques we will bring to bear will be those of logic and theoretical computer science, particularly synchronous process calculus [27, 28, 29].

Process calculi are formal systems that allow for the compositional construction of discrete dynamical systems. As such they may be fruitfully regarded as idealized languages for modelling and simulation. Furthermore, the use of a formal logic of system properties provides an analysis technique for systems described by the process calculus. We refer to this as the *two-language* approach, and it is greatly strengthened in the presence of automated model-checking.

All formal calculi make foundational commitments. Process calculi are no exception. Most standard process calculi make the commitment that system states are formally represented by a syntactic category of processes. Thus all aspects of system state must be represented within the syntax of processes: this includes any form of resource and location that one wishes to consider. The great advantage of this approach is that it allows one to work within a very elegant and minimal calculus, for which it is straightforward to prove metatheorems. On the other-hand, the disadvantage is that it can obscure the evolution of those features (location and resource) and add computational burden.

Pym and Tofts, and also Collinson, have initiated a line of research which explictly takes a contrary view [16, 17, 40, 41]. In the **SCRP** family of calculi, system states carry a resource

*matthew.collinson@hp.com

†brian.monahan@hp.com

‡david.pym@hp.com

§All authors: HP Labs, Stoke Gifford, Bristol BS34 8QZ, England, U.K.; David Pym also: University of Bath, BA2 7AY, England, U.K.

component and a process component, and we sometimes say that resources are first-class citizens within the formalism. The resource components are used to represent passive entities in models, whilst the process component is used, primarily, to represent active entities (agents).

The notion of resource (e.g., space, memory, money) taken corresponds to the resource semantics of bunched logic [34, 37, 38, 39], based on (ordered, partial, commutative) monoids (e.g., the non-negative integers with zero, addition, and less-than-or-equals), which captures the following basic properties of resources:

- Each type of resource is based on a basic set of resource elements;
- Resource elements can be combined (and the combination has a unit);
- Resource elements can be compared.

The notion of process is an algebraic one, based on a calculus that is a natural development in this setting of Milner's SCCS [28].

The basic idea is that resources, R, and processes, E, co-evolve,

$$R, E \xrightarrow{a} R', E',$$

according to the specification of a partial function, $\mu : (a, R) \mapsto R'$, that determines how an action a evolves E to E' and R to R'.

The base case of the operational semantics is given by action prefix:

$$\frac{}{R, a : E \xrightarrow{a} R', E} \qquad (\mu(a, R) = R').$$

Concurrent composition exploits the monoid composition on resources,

$$\frac{R, E \xrightarrow{a} R', E' \quad S, F \xrightarrow{b} s', F'}{R \circ S, E \times F \xrightarrow{ab} R' \circ S', E' \times F'}.$$

Sums and recursion are formulated in the evident way (see [17, 40, 41]; cf. Section 2). Of more interest is hiding,

$$\frac{R \circ S, E \xrightarrow{a} R' \circ S', E'}{R, (\nu S)E \xrightarrow{(\nu S)a} R', (\nu S')E'},$$

in which the resource S becomes bound to the process E. This construction replaces, and generalizes, the restriction operation of calculi such as SCCS.

The decomposition of states using resource makes the process calculus rather close in spirit to practical systems modelling languages, like Demos2k [21]. Further, it makes the states of the system amenable to the techniques of substructural, particularly bunched, logic [34, 37, 38, 39, 33]. The additional expressivity leads to logical characterizations of process constructors — a feature which is not usually present in two-language process calculi — and allows for the specification of the form of processes used in a model.

More specifically, with an explicit model of resources as above, we are able to work with a logical judgement

$$R, E \models \phi,$$

read as 'relative to the available resources R, process E has property ϕ'. In this setting, we obtain, using the multiplicative conjunction, $*$, a finer analysis of this logical judgement than is available in Hennessy-Milner logic [23, 45]. A characterization of parallel composition, $\times$, where $\sim$ is the appropriate notion of bisimulation, obtains as follows, where $\circ$ is resource combination and $\sqsubseteq$ is resource comparison:

$$\begin{array}{lll} R, E \models \phi_1 * \phi_2 & \text{iff} & \text{there are } R_1 \text{ and } R_2 \text{ such that } R_1 \circ R_2 \sqsubseteq R \\ & & \text{and there are } E_1 \text{ and } E_2 \text{ such that } E_1 \times E_2 \sim E, \\ & & \text{such that } R_1, E_1 \models \phi_1 \text{ and } R_2, E_2 \models \phi_2. \end{array}$$

That is, we are able to characterize logically the concurrent structure of the system, together with its resource-constrained synchronization. In addition to the usual (additive) modalities and quantifies of Hennessy-Milner logic, this set-up admits also multiplicative modalities and quantifiers (see [17, 40, 41]; cf. Section 6). A characterization of the hiding operator, similar to the one given above for product, is obtained using a multiplicative quantifier. This logic is called **MBI**.

If one wishes to construct models of systems with location in the **SCRP** calculus, then this must be done using the resource and/or process components. For some simple situations, the notion of location can be treated as resource — for example, this would be sufficient to give a process algebraic account of Separation Logic (where location is thought of as resource). For more complex notions, an encoding into the process component must be used.

The focus of this paper will be to extend the **SCRP** programme by having system states with three components: location, resource and process. Thus location will be treated as a first-class citizen in the formalism. This follows up a suggestion in [41] and completes the addressing of the structural components in our characterization of the conceptual components of a system model. (The remaining component, environment, is handled in our framework and tools using stochastic processes [51, 41, 6, 21].) We name the resulting (family of) calculi **LSCRP**.

The calculus gives rise to systems with dynamic behaviour of the form

$$L, R, E \xrightarrow{a} L', R', E'$$

where a is an action (in the usual process sense), L, L' are location environments, R, R' are resource environments and E, E' are processes used to control the evolution. Following the approach sketched above, we define a modification function, μ, which, for each action a, location L, and resource R, determines the evolved location L' and resource R'. In a state L, R, E, the L carries the relevant information about the topology of the model and R carries relevant information about the distribution of the resources around the model's topology.

In our subsequent development, the fragment consisting of resources and processes amounts to the calculus **SCRP** [16, 17, 40, 41], sketched above. For the purposes of this paper, we take as motivation the following as the properties of location that we wish to capture:

- A collection of atomic locations — the basic places — which generate a structure of locations;
- A notion of (directed) connection between locations — describing the distributed of the system;
- A notion of sublocation;
- A notion of substitution (of a location for a sublocation) that respects connections — substitution provides a basis for abstraction and refinement in out system models;
- A product of locations (an inessential but possibly useful technical property).

The notions of sublocation and substitution are intimately related, with the former being a prerequisite for the latter. We will not develop or implement substitution in this paper (except for brief comments in examples) rather deferring it as a next step.

Treating location as a first-class citizen in this way does not lead to a process calculus with operational behaviour that is more expressive in absolute terms. It does, however, lead to greater pragmatic expressiveness: we claim that it is simplifies the construction of models of a wide range of systems. It also makes it easier to write specifications about located resource in the logical language. In some circumstances, such as those that obtain in Separation Logic [43, 25], location can be treated as a form of resource. This is because, in such settings, the topology of locations essentially plays no role.

Our research in this area is partly driven by a number of practical systems modelling situations. The kinds of examples we consider are often quite detailed. In order to be able to tackle these examples efficiently, we have constructed Located Demos2k (LD2k) [12, 13, 14], a prototype tool for modelling with distributed resources. This tool is an extension of the existing Demos2k (D2k) tool. D2k has been used extensively in large-scale commercial projects undertaken by, and related to, HP's services businesses [46, 51, 3, 4]. Our intention is that LD2k should bear the same relation to **LSCRP** as D2k does to **SCRP**. In particular, **LSCRP** should be able to serve as a theoretical foundation for LD2k.

There are a number of alternative approaches to modelling with location in the literature of both logic and process algebra. For example, Barwise and Seligman [2] produce a logic for distributed systems based on translations along morphisms connecting domains used to model locations. The calculus of *(mobile) ambients* of Cardelli and Gordon [9] is a specialized calculus that combines process calculus with spatial structures. It provides processes that lie at the nodes of certain trees, and mobility corresponds to the ability of process terms to dynamically reconfigure such trees. An even more sophisticated system is the *bigraph* approach of Milner [26, 31]. As the name (perhaps) suggests, a bigraph is a mathematical structure that combines two types of structure. It has both a graph signifying connectivity and a graph signifying the sublocation relation. Milner's ingenious set-up allows for various algebraic connections and pluggings of such bigraphs, allowing for the compositional formation of complicated systems. These systems are equipped with rewrite rules, called reactions, that describe dynamical behaviour, in particular reconfgurations. Bigraphs have been shown to encode the above mobile ambients.

Our aims are somewhat more prosaic than the goals of bigraphs, and our destination may well be less powerful (in absolute terms of definability) than the calculus of mobile ambients. The simplicity of the underlying semantics leads, however, to (what we believe is) a particularly practical framework for a wide-range of real-world modelling problems, such as those described in [46, 51, 3, 4]. For the same reason, it comes equipped with a modal logic featuring natural assertions about location and resource.

In § 2, we give an introduction to the **LSCRP**-family of process calculi. In § 3, we study the metatheory of such calculi. In § 4, we give an introduction to the LD2k modelling language and a specialized version of **LSCRP** intended as an intermediary between the general theory of **LSCRP** and LD2k. In § 5, we give an example to illustrate the kind of idioms we wish to be able to easily express. In § 6, we give a logic for reasoning about **LSCRP** systems. We conclude with a discussion of our achievements and intended future work in § 7.

2 A General Theory of Location, Resource and Process

In this section, we define the **LSCRP**-family of process calculi. Each member of the family is determined by a signature consisting of supporting structures (for location and resource) and certain partial functions (to be explained). A number of choices have been made in the design of this version of **LSCRP**, and it is important to note that many other self-consistent choices are possible; we expect that some of these may occur to the reader. It should also be noted, however, that the theoretical development of these ideas is deceptively delicate, with many traps for the unwary. The apparently straightforward presentation given herein belies the delicacy in its development.

2.1 The Supporting Structures

We begin by defining mathematical structures to support our use of location and resource.

Each member of the family **LSCRP** is determined by a signature

$$(\mathsf{Act}, \mathbf{L}, \mathbf{R}, \mu, \nu)$$

consisting of the *action monoid*, *location structure*, *resource monoid*, *modification function* and *(action) hiding function*. Act generates the process terms and $\mathbf{R}$ is an ordered partial commutative monoid of rsources, as discussed above. Initially, we shall require just an order structure for $\mathbf{L}$ (to which a product can easily be added). Although connections are not (necessarily) represented explicitly, they are captured implicitly via the modification function, μ, which describes how resources move between locations (see below). Note that processes are not located in this sense.

The full name of each calculus is $(\mathsf{Act}, \mathbf{L}, \mathbf{R}, \mu, \nu)$-**LSCRP**, but will be referred to by the generic name **LSCRP** when the fixed signature is clear.

Partially defined expressions will be used a great deal. We write $E\downarrow$ or $E\uparrow$ when an expression E is either defined or undefined, respectively. We often use Kleene-equality, writing $E \simeq F$ if, E is defined exactly when F is defined and when defined they are equal.

A partial commutative monoid (or *pcm*) is a structure

$$\mathbf{R} = (\mathbf{R}, \sqsubseteq, \circ, \mathbf{e})$$

with carrier set $\mathbf{R}$ (which we do not distinguish from the structure), preorder $\sqsubseteq$, and partial binary composition $\circ$ with unit $\mathbf{e}$. This is further required to satisfy the *bifunctoriality condition*,

$$R \sqsubseteq R' \ \text{ and } \ S \sqsubseteq S' \ \text{ and } \ R' \circ S' \downarrow \quad \text{implies} \quad R \circ S \downarrow \ \text{and}\ R \circ S \sqsubseteq R' \circ S'$$

for all resources R, R', S, S' in $\mathbf{R}$. The natural numbers with their usual ordering, addition and zero provide a simple example of a pcm.

Fix the pcm $\mathbf{R}$ — this is to be used as the resource monoid. Reserve the letters R, S, T for the elements of the resource monoid $\mathbf{R} = (\mathbf{R}, \sqsubseteq, \circ, \mathbf{e})$, with *resource order* $\sqsubseteq$, *resource composition* $\circ$, and *resource unit* $\mathbf{e}$.

Fix a poset $\mathbf{L} = (\mathbf{L}, \preceq, \ell)$, with least element ℓ, to be used as the location structure. Reserve the letters L, M, N for elements of the location structure. The relation $\preceq$ is referred to as the *sublocation* relation.

The poset $\mathbf{L}$ is often a pcm, $\mathbf{L} = (\mathbf{L}, \preceq, \odot, \ell)$, but we shall not require this. The operation $\odot$ is then referred to as *location composition* and the location ℓ as the *location unit*. Given a location structure and resource monoid as above the set $\mathbf{LR} = \mathbf{L} \times \mathbf{R}$ itself carries an order

$$(L, R) \sqsubseteq (M, S) \qquad \text{iff} \qquad L \preceq M \ \text{ and } \ R \sqsubseteq S$$

for all locations L, M and resources R, S.

The location structure $\mathbf{L}$ and resource monoid give rise to a structure

$$\mathbf{LR}_{\circledast} = (\mathbf{LR}, \sqsubseteq_{\circledast}, \circledast, \{\mathbf{e}^L_{\circledast} \mid L \in \mathbf{L}\})$$

with

$$(L, R) \sqsubseteq_{\circledast} (M, S) \iff L = M \ \text{ and } \ R \sqsubseteq S$$

and

$$\begin{aligned} (L, R) \circledast (M, S) \quad &= \quad \begin{cases} (L, R \circ S) & \text{if } L = M \text{ and } R \circ S \downarrow \\ \uparrow & \text{otherwise} \end{cases} \\ \mathbf{e}^L_{\circledast} \quad &= \quad (L, \mathbf{e}). \end{aligned}$$

This is almost, but not quite, a pcm — there are no units, only a family of partial units $e^L_{\circledast}$, each of which is a unit for pairs with location component L.

In Proposition 3, below, we see that $\mathbf{LR}$ supports further useful structure of its own. If $\mathbf{L}$ is a pcm then $\mathbf{LR}$ can also be made into a pcm in the evident pointwise way — composition defined just when both component compositions defined — but we shall not make much use of this fact here. For notational convenience, we often omit the brackets around any pair $(L, R) \in \mathbf{LR}$.

The location structure and resource monoid will usually be very different. In particular, they will most often carry additional structure of very different kinds. Although this additional structure is not required for the general theory, it is often very important in particular situations. In contrast to the quantity-based resource monoid example above, the location structures used in practice are usually based upon structures with spatial characteristics, often graphs.

Example 1. The simplest practical examples of location structures are generated from some given set $\mathcal{V}$. The elements of $\mathcal{V}$ are referred to as *vertices*, *nodes* or *atomic locations*. Then, by using a powerset construction, we produce the carrier set of $\mathbf{L}$. For example, $\mathbf{L} = (\mathcal{PV}, \subseteq, \cup, \emptyset)$ is a pcm that can be used as a location structure.

Important variants include: the finite powerset; the powerset but with the non-overlapping union operation (composition undefined when arguments overlap); the hierarchy generated from $\mathcal{V}$ by iterations of the powerset. The second of these is important for situations in which we wish to make assertions about disjointness of locations, and the fourth for situations in which we have locations *containing* (embeddings of) sublocations. Notice that locations may overlap without either being a sublocation of the other. In this example, there are no connections that substitution must respect. $\square$

Example 2. (A topological example) A location $L = (X_L, \Omega_L)$ is a topological space. An atomic location is a point of L. The composite of locations is the sum of spaces. The sublocation relation is the subspace relation. The least element is the empty space. $\square$

Example 3. For a richer example, assume the existence of a directed graph, $\mathcal{G}$, with vertices $\mathcal{V}$ and uniquely labelled edges $\mathcal{E}$. Again, we think of the vertices as atomic locations. The edges are referred to as *basic links*. The unique labelling of edges is the assumption of an injective function from $\mathcal{E}$ to some set $\mathcal{N}$ of *(basic) link names*. We use the notation $l \overset{n}{\twoheadrightarrow} l'$ to signify a basic link with name n from vertex l to vertex l'. A location structure $\mathbf{L}$ can be generated as in the previous example, but using all subgraphs instead of all subsets. Thus a location, L, is just a subgraph of $\mathcal{G}$. The order between locations is the subgraph relation. Given a pair of subgraphs $\mathcal{G}^1 = (\mathcal{V}^1, \mathcal{E}^1)$ and $\mathcal{G}^2 = (\mathcal{V}^2, \mathcal{E}^2)$ we form the composite

$$\mathcal{G}^3 = (\mathcal{V}^3, \mathcal{E}^3) = \mathcal{G}^1 \circledcirc \mathcal{G}^2 = (\mathcal{V}^1 \cup \mathcal{V}^2, \mathcal{E}^1 \cup \mathcal{E}^2)$$

by taking the union of vertices and the union of the edges. The unit of this monoid is the empty graph.

Again, there are many useful variants of this basic version. It can also be useful to extend the basic links to *connections* between arbitrary (not necessarily atomic) locations. For example,

$$L \overset{n}{\twoheadrightarrow} L' \ \text{ iff } \ \exists l \in L. \exists l' \in L'.\, l \overset{n}{\twoheadrightarrow} l'$$

for all L, L' in $\mathbf{L}$ and $n \in \mathcal{N}$. These connections between arbitrary locations, which must be respected by substitution, provide a basis for defining, over this structure of locations, a modification function which respects the intended structural constraints of the model. $\square$

Example 4. We start with a labelled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of basic locations. The set $\mathbf{L}$ of locations is the disjoint union $\biguplus_{i \in \mathbb{N}} \mathbf{L}^i$ of sets $\mathbf{L}^i$ of graphs defined below. An element L of $\mathbf{L}^0$ is a subgraph of $\mathcal{G}$: we write $v \overset{n}{\twoheadrightarrow} v'$ when there is an edge from v to v' labelled by n. An element L of $\mathbf{L}^{i+1}$ is a graph such that each vertex is a set of elements of $\mathbf{L}^i$. Edges between vertices v and v' of $L \in \mathbf{L}^{i+1}$ are given by

$$v \overset{n}{\twoheadrightarrow} v' \ \text{ iff } \ \exists M \in v. \exists M' \in v'. \exists w \text{ a vertex of } M.\ \exists w' \text{ a vertex of } M'.\ w \overset{n}{\twoheadrightarrow} w'\ .$$

The order between locations is given for all L and M by: $L \preceq M$ iff there is some i such that $L, M \in \mathbf{L}^i$ and L is a subgraph of M. $\square$

Example 5. (Higher-order locations)

For any graph $\mathcal{G}$, let $\mathcal{C}(\mathcal{G})$ be the set of connected components. Assume a set V of atomic locations. A location L is any family of graphs $(\mathcal{G}^i_L = (\mathcal{V}^i_L, \mathcal{E}^i_L) \mid i \in \mathbb{N})$ satisfying:

$$\begin{array}{rcl} \mathcal{V}^0_L & \subseteq & V \\ x \in \mathcal{V}^{i+1}_L & \Longrightarrow & \exists X \subseteq \mathcal{C}(\mathcal{G}^i_L).\ x = \bigcup X \\ x, y \in \mathcal{V}^{i+1}_L & \Longrightarrow & x \text{ and } y \text{ are disjoint} \end{array}$$

where $\bigcup$ is the union of (disjoint) graphs. We call $\mathcal{G}^i_L$ the ith-*level* of L. The disjointness condition above ensures that no connected component at the ith-level can ever be included in two vertices at the $(i+1)$th-level.

The order on locations is

$$L \preceq M \quad \text{iff } \forall i \in \mathbb{N}.\ \mathcal{G}^i_L \subseteq \mathcal{G}^i_M$$

where $\subseteq$ is the subgraph relation. The least element ℓ is the location which is empty at all levels.

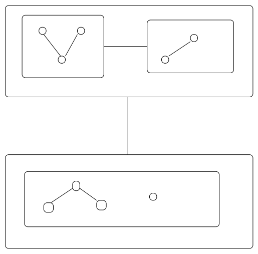
Let $L = (\mathcal{G}^i_L \mid i \in I)$ and $M = (\mathcal{G}^i_M \mid i \in)$ be locations. Define the compositie $L \odot M$ as follows:

$$L \odot M \downarrow \qquad \text{iff} \qquad V^0_L \cap V^0_M = \emptyset$$

$$\mathcal{G}^i_{L\ \odot\ M} = \mathcal{G}^i_L \cup \mathcal{G}^i_M$$

for each $i \in \mathbb{N}$, where note that $\cup$ is the (disjoint) union of graphs.

Note that L may be such that $\mathcal{G}^i_L$ is empty for all sufficiently large i. An example of such an L is shown below.



Pictures of this kind are often found to be a convenient level of representation in discussions of distributed systems. □

As in other process calculi, processes are generated from the actions that they may perform. An *action* is an element of the given monoid Act. This is not required to be a resource monoid, nor is it required to be a group (as it is in SCCS), but the monoid composition is required to be total. We write composition as juxtaposition (occasionally using brackets for clarity) and write the *unit* (also known as *tick*) as 1. Let a, b, c range over actions. In most cases, the monoid is freely generated from a countable set Act_0 of *atoms* — we reserve the letter α for atomic actions.

Given a choice of Act, $\mathbf{L}$ and $\mathbf{R}$ a *modification* is a partial function

$$\mu : \mathsf{Act} \times \mathbf{L} \times \mathbf{R} \rightharpoonup \mathbf{L} \times \mathbf{R}$$

satisfying the conditions:

(id) $\mu(1, L, R) = L, R$

($\otimes$**c**) if $\mu(a, L, R)$, $\mu(b, L, S)$ and $R \circ S$ are defined then

$$\mu(ab, L, R \circ S) \simeq \mu(a, L, R) \circledast \mu(b, L, S)$$

for all locations L, M, resource R, S and all actions a, b. These are sometimes referred to as the *identity* and *located composition* conditions.

Sometimes a variant ($\otimes$**sc**) of ($\otimes$**c**) is required in which the Kleene-equalities ($\otimes$**c**) is taken to be actual equality, that is, the quantities on either side are defined. The following additional conditions are sometimes required:

(lm) If $\mu(a, L, R)\downarrow$ and $L \preceq M$ then $\mu(a, M, R)\downarrow$ and $\mu(a, L, R) \sqsubseteq \mu(a, M, R)$

(re) If $\mu(a, L, R) = (L', R')$ and $R \circ S \downarrow$ then $\mu(a, L, R \circ S) = (L', R' \circ S)$.

If μ satisfies (**id**), (**$\otimes$sc**), (**lm**) and (**re**) then we say that it is a *strong* modification.

It is often useful to write the modification using a pair μ_1, μ_2 of functions, so that

$$\mu(a, L, R) = \mu_1(a, L, R), \mu_2(a, L, R)$$

for all actions a, locations L and resources R. In many applications either or both of these functions is independent of the other. In constructing models, it is intended that μ respects the spatial structure of locations (often given by the connections).

A *hiding* is a (total) function

$$\nu : \mathbf{L} \times \mathbf{R} \times \mathsf{Act} \longrightarrow \mathbf{R}$$

with no particular properties required. We write the result of hiding R at L in a as $\nu(L, R)a$. For example, an important choice of hiding is

$$\nu(L, R)a = \prod_{1 \le i \le m \ \& \ \mu(\alpha_i, L, R)\uparrow} \alpha_i$$

for all a, L and R, where Act is freely generated and a is written as a product of atoms $a = \alpha_1 \ldots \alpha_m$ for some integer $m \ge 0$. Note that if $m = 0$ then the above product defines the tick action 1. The hiding is used to give a resource-based form of restriction in the process calculus.

2.2 The Process Calculus

In addition to the data $(\mathsf{Act}, \mathbf{L}, \mathbf{R}, \mu, \nu)$ we further assume a collection of *(process) variables*, ranged over by X. Processes are formed according to the grammar

$$E ::= \quad X \mid a : E \mid \textstyle\sum_{i \in I} E_i \mid \nu(R)E \mid \mathit{fix}_j \mathsf{X}.\mathsf{E} \mid E \otimes E,$$

where X is a process variable, a is an action, I is any set, $L \in \mathbf{L}$, $R \in \mathbf{R}$, $1 \le j \le m$, X is an m-tuple of process variables, and E an m-tuple of processes. We refer to these forms as *variables*, *prefixes*, *sums*, *hidings*, *fixed points* and *(located) products*, respectively. We use brackets to disambiguate process expressions.

Write E_i for the ith component of any tuple E of processes. The expression $\mathit{fix}_j \mathsf{X}.\mathsf{E}$ is an alternative notation for $(\mathit{fix}\mathsf{X}.\mathsf{E})_j$. We often use the infix notation $+$, rather than summation notation, for binary sums. The *zero* process, $\mathsf{0}$ is formed by taking a sum indexed by the empty set. When $m = 1$ for the m-tuples in a fixed point we drop the subscript in the fixed point notation and write $\mathit{fix}X.E$ instead of $\mathit{fix}_1 \mathsf{X}.\mathsf{E}$, where X and E are the components of X and E, respectively. The unit process $\mathsf{1}$ is defined by taking the fixed point $\mathsf{1} = \mathit{fix}X.1 : X$. In the usual way, fixed points may be defined equivalently by systems of simultaneous recursion equations. For example, $\mathsf{1} := 1 : \mathsf{1}$ is an alternative way to define the unit process.

In the usual way, the process variables X are bound in a fixed point $\mathit{fix}_j \mathsf{X}.\mathsf{E}$. We distinguish *open* processes which contain free process variables and *closed* processes which do not. We write $\mathsf{E}[\mathsf{F}/\mathsf{X}]$ for the n-tuple of processes formed by substituting each F_i for each X_i into the n-tuple E, where F is an m-tuple of processes and X is an m-tuple of variables. Let Proc be the set of all processes and CProc be the set of closed processes. A *state* is a triple L, R, E, where L is a location, R is a resource and E is a process. A state is closed just when its process component is closed. Let States be the set of states and $\mathsf{CStates}$ be the set of closed states.

The dynamic behaviour of processes uses a state-space with states of the form

$$L, R, E,$$

where $L \in \mathbf{L}$, $R \in \mathbf{R}$ and E is a process. For notational convenience we identify triples L, R, E with pairs $(L, R), E$. The dynamics are given as a transition system, with a family of action-labelled binary transition relations, with transitions of the form

$$L, R, E \xrightarrow{a} L', R', E'$$

Prefix
$$\frac{}{L, R, a : E \xrightarrow{a} \mu(a, L, R), E}$$

Sum
$$\frac{L, R, E_i \xrightarrow{a} L', R', E'}{L, R, \sum_{i \in I} E_i \xrightarrow{a} L', R', E'}$$

Fix
$$\frac{L, R, \mathsf{E}_i[\mathsf{E}/\mathsf{X}] \xrightarrow{a} L', R', E'}{L, R, \mathit{fix}_i \mathsf{X}.\mathsf{E} \xrightarrow{a} L', R', E'}$$

L-Product
$$\frac{M, R, E \xrightarrow{a} M', R', E' \qquad N, S, F \xrightarrow{b} N', S', F'}{L, R \circ S, E \otimes F \xrightarrow{ab} \mu(ab, L, R \circ S), E' \otimes F'} \quad (S\otimes)$$

Hide
$$\frac{L, R \circ S, E \xrightarrow{a} L', R' \circ S', E'}{M, R, \nu(S)E \xrightarrow{\nu(L,S)a} \mu(\nu(L, S)a, M, R), \nu(S')E'} \quad (S\nu)$$

Frame
$$\frac{L, R, E \xrightarrow{a} L', R', E'}{L, R \circ S, E \xrightarrow{a} L', R' \circ S, E'} \quad (\mu(a, R \circ S)\downarrow)$$

Figure 1: Structural Operational Semantics for **LSCRP**

between states. This family is specified uniquely by the structural operational semantics given in Figure 1 below. Thus, it is the smallest family of transitions satisfying the rules. There are implicit side-conditions that all values and terms appearing in the rules are defined — for example, in the Prefix rule it is implicit that $\mu(a, L, R)$ is required to be defined for the indicated transition to exist. The explicit side-conditions in Figure 1 are:

$$\begin{array}{ll} (S\otimes) & M \preceq L \text{ and } N \preceq L \\ (S\nu) & L \preceq M \text{ and } \mu_1(\nu(L, S)a, L, R) = L'. \end{array}$$

This completes the definition of an **LSCRP**-calculus. The sub-family of **OLSCRP**-calculi consists of those with a strong modification.

The calculi we have set up above emphasize the ordered structure of location. An alternative choice is to emphasize the partial monoidal structure. This leads to a different choice of synchronous product rule. In particular, one can study calculi in which location is treated in the same way as resource: that is, the location environment is split by the processes in a synchronous product. It is also possible to consider products in which the components are constrained to either start or finish in the same location. The Frame rule is recognizable as a structural rule in the usual sense of proof theory.

2.3 Simulation

In a process algebra, one should have a notion of equality for process terms that goes beyond syntactic identity and identifies processes with similar behaviour. The usual notion is that of an equivalence relation called *bismulation.* The treatment in **SCRP**-like calculi is rather subtle because of the decomposition of state. The addition of location has been set up so as to be a straightforward adaptation of the underlying set-up for **SCRP**.

We define the *local equivalence* relation $\approx \subseteq \mathsf{CStates} \times \mathsf{CStates}$ to be the largest binary relation on closed states such that the following condition holds: if there are any two states in the relation $L, R, E \approx M, S, F$ then

1. $L = M$ and $R = S$,
2. if there is a transition $L, R, E \xrightarrow{a} \mu(a, L, R), E'$ for any a and E' then there is a some transition $L, R, F \xrightarrow{a} \mu(a, L, R), F'$ with $\mu(a, L, R), E' \approx \mu(a, L, R), F'$ for some F', and
3. if there is a transition $L, R, F \xrightarrow{a} \mu(a, L, R), F'$ for any a and F' then there is some transtion $L, R, E \xrightarrow{a} \mu(a, L, R), E'$ with $\mu(a, L, R), E' \approx \mu(a, L, R), F'$ for some E'.

The local equivalence relation is extended to a relation $\approx \subseteq \mathsf{States} \times \mathsf{States}$ on all states by defining $L, R, E \approx L, R, F$ iff $L, R, E[\mathsf{E}/\mathsf{X}] \approx L, R, F[\mathsf{E}/\mathsf{X}]$ for all m-tuples E of closed processes, where X is an m-tuple containing (one-copy of) all free variables of E and F.

Define $E \approx F$ iff $\forall L. \forall R.\ L, R, E \approx L, R, F$, and $\mathsf{E} \approx \mathsf{F}$ iff $\forall 1 \le i \le n.\ \mathsf{E}_i \approx \mathsf{F}_i$ for processes E, F and n-tuples of processes E and F. We say that two processes E and F are *locally equivalent* whenever $E \approx F$ holds.

The local equivalence on states looks very like the standard notion of bisimulation. It is not, however, a congruence [16]: it is truly local. Even in the absence of the congruence property, it remains a useful relation.

We sometimes use a more restrictive relation which is a congruence. This time the relation is defined initially on processes rather than states. Define the *global equivalence* relation $\sim \subseteq \mathsf{CProc} \times \mathsf{CProc}$ to be the largest binary relation on closed processes such that: if any two such processes are in the relation $E \sim F$ then

1. if there is any transition $L, R, E \xrightarrow{a} \mu(a, L, R), E'$ for some L, R, a and E', then there is some F' with a transition $L, R, F \xrightarrow{a} \mu(a, L, R), F'$ and $E' \sim F'$, and
2. if there is any transition $L, R, F \xrightarrow{a} \mu(a, L, R), F'$ for some L, R, a and F', then there is some E' with a transition $L, R, E \xrightarrow{a} \mu(a, L, R), E'$ and $E' \sim F'$.

This is extended to all processes (not just closed ones) and to tuples of processes in the usual way (as with $\approx$ above). It is then extended to states, by taking $L, R, E \sim L, R, F$ just when $E \sim F$, for and L and R. The remainder of this paper will use the global equivalence.

3 Properties of the General Theory

In this section, we explore some of the technical properties of the calculi.

3.1 Transition Properties

The transition structure can be restricted to the closed states, since they are closed under transitions.

Lemma 1. If L, R, E is a closed state and $L, R, E \xrightarrow{a} L', R', E'$ is any transition, then L', R', E' is a closed state.

Proof. By induction on the derivation of transitions. The Prefix case is immediate. The Sum, L-Product, Product, Hide, Fix and Frame cases all follow straightforwardly using the evident induction hypothesis. □

The evolution of processes is, in general, non-deterministic as the Sum rule allows for choice between many actions, and there can be multiple decompositions of resource in the L-Product and Hide rules. However, once the action has been chosen, the evolution of resource is completely deterministic.

Lemma 2. If $L, R, E \xrightarrow{a} L', R', E'$ then $L', R' = \mu(a, L, R)$.

Proof. By induction on derivations. The Prefix case is by definition. The Sum, Fix and Frame cases are by the evident induction hypothesis. The Product, L-Product and Hide cases use the side-conditions on the rules. The result then follows since the family of transition relations is the smallest such family that is closed under the rules. □

Lemma 3. If $L, R, E \xrightarrow{a} \mu(a, L, R), E'$ and $L \preceq M$ then $M, R, E \xrightarrow{a} \mu(a, M, R), E'$ is also a transition, provided μ is a strong modification.

Proof. By induction on the derivation of transitions according to the SOS and, therefore, by case analysis of rule instances. In the case of Prefix, the condition (**lm**) gives the result. The Sum, Fix and L-Product cases are simple applications of the induction hypothesis. The Hide case is immediate since any preorder $\sqsubseteq$ is transitive. □

Thus there is an admissible structural rule

$$\frac{L, R, E \xrightarrow{a} \mu(a, L, R), E'}{M, R, E \xrightarrow{a} \mu(a, M, R), E'} \quad (L \preceq M)$$

valid in **OLSCRP**-calculi.

Proposition 1. The global equivalence relation $\sim$ on states is contained in the local equivalence relation $\approx$.

Proof. It is easy to verify that $\{\langle (L, R, E), (M, S, F)\rangle \mid L, R, E \sim M, S, F\}$ is a set of pairs of states which is closed under transitions in the sense required by the definition of $\approx$. Thus $\sim$ is smaller than $\approx$. □

The converse to the above proposition does not hold, and the counterexamples from [16, 17] can easily be modified to show this.

3.2 Simulation Results

Here we present some of the algebraic theory of location, resource and process.

Proposition 2. The relation $\sim$ on processes is a congruence with respect to the process constuctors. To be precise, it is an equivalence relation and satisfies:

$$\begin{array}{cc} a : E \sim a : F & E + G \sim F + G \\ \nu S.E \sim \nu S.F & E \otimes G \sim F \otimes G \\ \mathit{fix}_i \mathsf{X}.\mathsf{E} \sim \mathit{fix}_i \mathsf{X}.\mathsf{F} & \end{array}$$

for all actions a, processes G, resources S and $E \sim F$ and $\mathsf{E} \sim \mathsf{F}$. Similarly, the relation $\sim$ is a congruence.

Proof. Mostly a straightforward verification using the standard methods. As usual, the clause for *fix* uses the preceding clauses, and an induction to show that $L, R, G[\mathit{fix}\mathsf{X}.\mathsf{E}/\mathsf{X}] \approx L, R, G[\mathit{fix}\mathsf{X}.\mathsf{F}/\mathsf{X}]$ for all L, R, G. The proofs for $\sim$ are only slightly different. □

Various simple algebraic properties hold for processes in **OLSCRP**.

Lemma 4. With $\approx$ as equality between **OLSCRP**-processes:

$$\begin{array}{cc} E + F \sim F + E & E + (F + G) \sim (E + F) + G \\ E + 0 \sim E & \\ E \otimes 0 \sim 0 & E \otimes 1 \sim E \\ E \otimes F \sim F \otimes E & E \otimes (F \otimes G) \sim (E \otimes F) \otimes G \end{array}$$

The same properties hold with $\sim$ in place of $\approx$.

Proof. The proofs are all by the standard method — checking that relations are closed under all possible transitions are required. The unit property for $\otimes$ requires the fact that ℓ is the least element of L, and the Frame rule. $\square$

It is not, in general, possible to represent a process E by a globally equivalent normal form $\mathcal{N}(E)$ consisting of a sum of prefixes of the immediate actions of E, because of the possibility of multiple decompositions of the same resource. Therefore the standard method for proving completeness of the equational theory of bisimulation cannot be applied to $\sim$ in the current, general setting.

The above facts allow us to conclude the following proposition, in the presence of the Frame rule.

Proposition 3. The state space (quotiented by the congruence $\sim$) is a disjoint union of pcms indexed by locations L. Define composition on the whole state space by Kleene-equality

$$(L, R, E) \circ (M, S, F) = \begin{cases} (L, R \circ S, E \otimes F) & \text{if } L = M \\ \uparrow & \text{otherwise} \end{cases}$$

and pre-order by

$$(L, R, E) \sqsubseteq (M, S, F) \quad \iff \quad L \preceq M \text{ and } R \sqsubseteq S \text{ and } E \sim F$$

for all L, M, R, S, E, F. The composition has a family of units

$$\mathbf{e}_L = (L, \mathbf{e}, 1)$$

indexed by locations L.

4 A Basic Model, a Special Theory, and a Modelling Tool

It is not always easy to write and comprehend models of large systems in a process algabraic formalism, even when an interpreter exists for automated execution. It is often easier to work in a higher-level simulation language. However, most such languages lack a clear and coherent semantic foundation.

The existing tool D2k combines the advantages of both approaches as one may write models in a higher-level language, but the tool has a semantics in a variant of SCCS (it is also rather close to **SCRP**).

In this section, we give an introduction to LD2k, an extension of D2k for more easily writing models with distributed resource and evolving spatial structure. It is intended that this language be founded in a particular member of the **LSCRP**-family, which we describe here. So far, this correspondence is conceptual rather than formal. That is to say, the prototype tool embodies essentially the same notion of location, resource and process. However, the very considerable work of giving a semantics to LD2k in **LSCRP** has not been completed. We give an extended example in both LD2k and **LSCRP**-syntax to highlight the close correspondence betwen the two.

4.1 LD2k

The general principles behind the LD2k tool [12, 13, 14] are already present in the existing D2k tool [21]. Our development makes one extension/refinement of the tool, namely the explicit treatment of location as a first-class citizen. These commitments are motivated by a modelling framework with the following four essential features:

Environment: We consider that the environment is the source of events that are incident upon the system's (logical or spatial) boundary. The events considered include not only the intended interaction of the system with its environment, but also unintended interactions, such as security incidents. Mathematically, we consider these events to be represented stochastically;

Locations: In general, a system is not confined to a single (logical or spatial) place. Rather, it is distributed over a collection of interconnected places. In the current version of LD2k, one may declare (atomic) locations and connections between them. This is closely related to Example 3 above. In particular, location is used to house resource (rather than process) as discussed below. At any point in the execution of a model there is a unique location environment (graph) associated. Location environments form a location structure: there is a partial order and a least element. They also form a pcm in a natural way. There is an evident notion of connection-preserving substitution, although this is yet to be exploited;

Resources: Resources are, essentially, the passive components of the system. They are both required for processes to execute and manipulated by process execution. In the current version of LD2k all resources are associated with an atomic location. Thus, at any stage of the execution of a model there is a unique resource environment associated. Resource environments can be combined and compared. Mathematically, we require that the set of resource environments carries the structure of a pcm;
Processes: The processes that execute (relative to the resources available around the distributed system) describe the dynamics of the system and the service it provides. Mathematically, we describe processes as terms of a process algebra. More specifically, we use a synchronous calculus of processes which execute relative to available resources. Note that processes are not themselves associated directly with locations.

LD2k models contain an initial segment in which various declarations are made — let us call this the *preamble*. The atomic locations and connections present in the model are declared in this preamble. For example,

```
location loc1;
location loc2;
connect loc1 -> loc2;
```

declares two locations and a link from the first to the second. Resources are then declared at various of these locations. For example,

```
newR(res1@loc1 ,4);
newR(res1@loc2 ,5);
newR(res2@loc2 ,3);
```

declares four units of resource $res1$ at location $loc1$, etc. Note that the same sort of resource may be declared at more than one location

The LD2k syntax allows for the construction of simple concurrent processes from a small collection of basic actions (there are also atomic imperative assignments, but we do not wish to complicate the discussion unnecessarily at this point). Here we focus on the core actions that deal with location and resource. These take the forms shown in Figures 2 and 3.

A simplifying design decision was to keep location-modifying and resource-modifying actions separate. Notice also that the resources currently present in LD2k have an implicit protocol attached: processes cannot put what they do not own, but must return all resources before they terminate. There are other types of resource, *bins* and *syncs*, in D2k, which we intend to build into LD2k in the near future. The forget and recall actions were not implemented in the prototype version in [12]. We have since extended the prototype with these features [14].

Atomic actions are assembled into processes by standard imperative programming constructs: sequencing, conditionals and while loops. An example process declaration is of the form:

```
process Proc1 = { ... }
```

where the imperative ‘programs’ are inserted between the braces. During the execution of a model, individual process instances are then launched at chosen intervals. For example,

```
launch(Proc1 ,t)
```

getR(r@l, n)	Get n units of resource r from location l, provided they are available, otherwise wait until they are, and then get them. These resources are then owned by the process issuing this action.
putR(r@l, n)	Put n units of resource r at location l, provided the process owns at least n units assigned to this location, otherwise wait until they are, then perform this action. After this action the process issuing this action owns n fewer of the resource at this location.
moveR(r, l −> m, n)	The issuing process waits until it owns n units of resource from location l. When it does so, it transfers them to location m. Afterwards, the additional n units are owned by the process at the location m.

Figure 2: LD2k resource actions

forget(l −> m)	The issuing process instructs the location environment to drop the link from location l to location m. If the link is already absent, then skip.
recall(l −> m)	The issuing process instructs the location environment to recall the link from l to m. The link must have been declared in the preamble. If it is already present, then skip.
hold(t)	Delay the issuing process for t units of time.

Figure 3: LD2k location and timing actions

launches an instance of Proc1 after time delay t.

To get a better feel for the power of the language and how it is intended tobe used, it may be instructuve to turn briefly to examine the lengthy example in Subsection 5.2.

4.2 The Basic Location Model and the Special Theory

The basic model and particular process calculus that follow have been used extensively to inform the design of LD2k.

In LD2k, all of the underlying location structure is initially declared in the preamble (before the dynamical behaviour is initiated, and outside of the process definitions), via the location and connect declarations. Semantically, this amounts to assuming the existence of a given graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of atomic locations and basic links, as in Example 3 above, but with unlabelled edges.

Let the location monoid $\mathbf{L}$ be generated by taking all finite subgraphs of $\mathcal{G}$. A *location (environment)* is a such a finite subgraph. We use the restriction of the location composition (union on both vertices and edges), unit (empty graph) and sublocation (subgraph) from Example 3.

Resources in LD2k are initially declared in the preamble using the newR declaration. However, the quantities, locations and availability of resources vary dynamically during execution. A declaration newR(r@l, Q) creates a resource r at location l with initial contents Q. Define r to be a *resource name* and let the set of all such names be RNm. The *quantity* Q is drawn from a fixed, total resource monoid $\mathbf{R}_r = (\mathbf{R}_r, \sqsubseteq_r, \circ_r, \mathbf{e}_r)$ with subtraction. That is, for all Q_1, Q_2 in $\mathbf{R}_r$, there is at most one Q_3 such that $Q_1 = Q_2 \circ_r Q_3$. We write this Q_3 as $Q_1 - Q_2$. Examples are the positive integers (as in current LD2k) or rationals with addition as the monoid composition. For the current version

$$\mathbf{R}_r = \mathbb{N} = (\mathbb{N}, =, +, 0)$$

for all resource names r. We call $\mathbf{R}_r$ the *sort* of r, and assume that it is unique: thus one cannot, in the same model, declare newR(r@l, Q) and newR(r@l′, Q′) with Q and Q' of different sorts.

Eventually, non-commutative sorts $\mathbf{R}_r$ will be required for dealing with value-bins, basically lists with composition as concatenation, but in this paper we ignore this complication.

A *resource (environment)* R is a dependent function with finite support from $\mathcal{V} \times RNm$ to quantitities of the appropriate sort,

$$R \in \prod_{(l,r)\in\mathcal{V}\times RNm} \mathbf{R}_r \ .$$

That is, for any $(l, r) \in \mathcal{V} \times RNm$ we have $R(l, r) \in \mathbf{R}_r$, and furthermore $R(l, r) = e_r$, the unit of $\mathbf{R}_r$, for all but finitely many (l, r). The set of all such resources gives the carrier of the resource monoid $\mathbf{R}$. The composite of resources R and S is defined pointwise as follows:

$$(R \circ S)(l, r) = R(l, r) \circ_r S(l, r)$$

for all (l, r). The order on $\mathbf{R}$ is the discrete order,

$$R \sqsubseteq S \quad \text{iff} \quad R = S$$

for all resources R, S. The unit of the resource monoid is $\mathbf{e_R}$ with $\mathbf{e_R}(l, r) = \mathbf{e}_r$ for all (l, r).

Write $R_{(l,r,Q)}$ for the resource which maps (l, r) to Q and all other (l, r) to the appropriate unit. Note that $\mathbf{R}$ has subtraction, and we use the notation $R - S$ for the (possibly undefined) term representing the resource formed by subtracting S from R. Note that $\mathbf{L}$ also has subtraction, and extend the notation in the evident manner. Write $L_{(l,l')} = (\{l, l'\}, \{l \twoheadrightarrow l'\})$ for the subgraph of $\mathcal{G}$ consisting of the single link $l \twoheadrightarrow l'$, and note that $L_{(l,l')} \downarrow$ iff $l \twoheadrightarrow l' \in \mathcal{E}$.

We fix a small number of atomic actions for use as the primitives of our process language. They are as follows

$\mathsf{get}(l, r, Q)$	get quantity Q of resource r from node l
$\mathsf{put}(l, r, Q)$	put quantity Q of resource r at node l
$\mathsf{move}(r, l, l', Q)$	shift quantity Q of resource r from location l to l'
$\mathsf{forget}(l, l')$	drop the underlying link $l \twoheadrightarrow l'$
$\mathsf{recall}(l, l')$	pick-up the underlying link $l \twoheadrightarrow l'$.

All actions are assumed to be generated from these atoms.

Thus there are two sharply distinguished types of atomic action:

- those that modify the resource distribution, but leave the available location structure untouched;
- those that modify the available location structure, but leave the resource alone.

The behaviours of actions are described (using Kleene-equality) by the modification μ given on atomic actions in Figure 4.2.

The proof of the following Proposition is then entirely standard, with the key fact being the existence of subtraction on the location and resource monoids.

Proposition 4. The assignments above for μ specify a unique modification.

The resulting modification is strong, so we generate a member of the **OLSCRP**-family of Section 2 with processes E and states L, R, E.

A state

$$L, R, E$$

now contains the following information:

- E, a process term, used to control transitions;
- L, the current location environment, determining which parts of $\mathcal{G}$ that E may (currently) use;

$$\begin{array}{rcll}
\mu(\mathsf{put}(l,r,Q),L,R) & = & L, R \circ R_{(l,r,Q)} & \\
\mu(\mathsf{get}(l,r,Q),L,R) & \simeq & L, R - R_{(l,r,Q)} & \\
\mu(\mathsf{move}(r,l,l',Q),L,R) & \simeq & \begin{cases} L, (R - R_{(l,r,Q)}) \circ R_{(l',r,Q)} & \text{if } l \twoheadrightarrow l' \\ \uparrow & \text{otherwise} \end{cases} & \\
\mu(\mathsf{forget}(l,l'),L,R) & \simeq & L - L_{(l,l')}, R & \\
\mu(\mathsf{recall}(l,l'),L,R) & \simeq & L \odot L_{(l,l')}, R &
\end{array}$$

Figure 4: Specification of LD2k modification

- R, the current resource environment, determining which resources are (currently) available to E, and at which atomic locations, $l \in L$.

The three state components co-evolve and there are often strong dependencies between all of them. For example, we have processes E with actions that move resource r from some atomic location l to another l': in such examples the location L must contain some connection $l \twoheadrightarrow l'$, and R must contain (l, r, Q) with sufficiently large Q. We also have processes that alter the visible sub-graph L of $\mathcal{G}$: for example, some link between atomic locations may be broken, thus preventing future move actions. On the other-hand, the breaking of this link may depend upon some permission held as a resource r at some atomic location.

Partial definedness is used to represent the inability of a process to perform an action at a particular time. Thus processes which repeatedly try to perform an action are represented using sums and fixed-points. The actions get, put and move act on a slightly more general resource than the particular LD2k-style resource. Processes do not own resources after such actions and there are no associated protocols relating to such ownership. Ownership is only for the duration of the corresponding actions. The move action actually moves the specified quantity of resource, rather than shifting the obligations as to which quantities resources are to be returned at the various locations.

4.3 Comparison of LD2k with the Special Theory

As LD2k is substantially based upon the modelling language D2k, it naturally inherits a number of characteristics such as

- the real-valued passage of time,
- imperative actions/effects for a guarded-command like language,
- shared variables, and
- both sequential and synchronously concurrent processes.

The mathematical semantics of applied languages like LD2k are, ideally, naturally expressed as a straightforward homomorphism from an appropriate term algebra representing the language itself into an algebra expressed in terms of a process algebra (here **LSCRP**). Our particular collection of language features represents, however, an interesting challenge to capture directly in this way. Our purpose here is to mention briefly some of these technical challenges, and then to point towards how we believe they can be resolved.

4.3.1 Sequencing and timed actions

Each LD2k process is internally sequential, consisting of both simple and compound statements. Simple statements, for example, consist of simple assignments and 'holds' in which an explicit amount of (real-valued) time passes. Other processes can, however, be launched after an explicit (real-timed) time delay. Resources can be claimed and released. Compound statements include scoping block statements, looping constructs such as repeat and while and finally the try conditional statement.

The 'try' statement inherited from D2k is particularly useful since it effectively provides multi-way synchronisation of resource access and introduces both blocked waiting and non-blocking tests. Because we have both simple and compound statements in LD2k, this means that sequencing — the semi-colon operator — would not directly correspond to an Action Prefix (or even the asynchronous action prefix) in **LSCRP**. As is well-known, explicit acknowledgement of completion is required to deal with sequencing. The **SCRP** semantics of a simple imperative language has been sketched in [16].

Now, each LD2k process moves forward in timed synchrony - that is, real-valued time passes simultaneously in each process. The upshot of this is that each statement may concurrently *consume* different amounts of time. We can illustrate this as follows:

```
process A = { hold(1.4); print("A - 1");
              hold(3.4); print("A - 2"); }

process B = { hold(0.8); print("B - 1"); hold(14.0);
              print("B - 2"); hold(2.0); }

launch("process-A", A, 1.0); // Launches instance of process
                             // A after 1.0 unit of time.
launch("process-B", B, 0.3); // Launches instance of process
                             // B after 0.3 units of time.

hold(37.4);
close;
```

Executing this fragment in LD2k results in an explicit timed trace:

```
 0.0  :  simulation begins
 0.3  :  "process-B-1" starts
 1.0  :  "process-A-1" starts
 1.1  :  "process-B-1" prints "B - 1"
 2.4  :  "process-A-1" prints "A - 1"
 5.8  :  "process-A-1" prints "A - 2"
 5.8  :  "process-A-1" exits
15.1  :  "process-B-1" prints "B - 2"
17.1  :  "process-B-1" exits
37.4  :  simulation closes
```

Notice that both the instances of process A and process B make progress — even though each action they perform happens at a different moment in time.

The form of synchrony covered here in **SCRP** (given by the L-Product rule in Figure 1) assumes that each action is atomic and synchronization take place at the completion of each action. In the case of LD2k, we must mark the flow of time even though some of the actions in each component process may not have completed.

4.3.2 Starting and stopping processes synchronously

Processes in LD2k can take different amounts of elapsed time to complete. Thus, LD2k processes can start and terminate at different times. Obviously this is awkward to do with a synchronised product that proceeds in lock-step. We need to be able to introduce and eliminate LD2k processes cleanly without unduly disrupting the execution of other, independent processes.

Fortunately, termination can be cleanly handled in a translation of each terminating LD2k process into **LSCRP** so that the translated process always ends in the process 1, so that we can exploit the fact:

$$1 \otimes E \sim E \sim E \otimes 1$$

for all processes E.

Dually, starting a fresh process instance can exploit the same idea (but in reverse) to *introduce* process terms into a product. In practice, we may also need to have appropriate process creation and termination rules to expand and contract synchronous product terms.

4.3.3 Tracking ownership of shareable resources

Like its predecessor D2k, LD2k keeps track of how much sharable resource that each process has claimed. The idea is to guarantee that, when a process terminates, it has exactly returned all of this resource, or else that the simulation aborts when that is not the case. This is a highly useful integrity property for LD2k processes to have automatically and systematically checked.

This tracking of resource ownership is not naturally incorporated within **LSCRP** — but it could be handled as part of a semantic translation from LD2k terms into **LSCRP** terms. What is required here is suitable mechanism to maintain a per-process dynamic mapping of how much resource each process has claimed, and to check at process termination that this mapping is empty.

4.3.4 Partiality of actions

The concept of location in **LSCRP** introduces more ways of naturally expressing distinctions into **SCRP** models. Consequently, this means that LD2k can also introduce more ways of naturally expressing and introducing distinctions. This also means, however, that functions, actions, and decisions can depend upon these distinctions — and, in particular, they may also fail. For example, access control decisions typically depend upon the identity/role of the claimant, their location, and for what purpose they are claiming the resource.

The upshot here is that incorporating location into LD2k introduces the potential for partially defined actions. This is a strict extension of the D2k execution model — if a resource is not currently available in D2k, then it is assumed that resource may be available at a later stage — there is no explicit *refusal* to provide resource in D2k.

This is of course different in LD2k and thus we need to represent how to manage such refusals when they arise. Incorporating this will require both syntactic and semantic extensions to the current LD2k framework.

4.3.5 Priority and choice

As mentioned above, a significant feature of both D2k and LD2k is the try statement; here is an example:

```
try [ getR(ports@server42, 1), count < 100 ] with {

  ... do statements (block 1) ...

}
etry [ getR(fileShares@server67, 4) ] with {

  ... do other statements (block 2) ...

}
etry [ getR(account@client35, 4),
       getR(writePrivilege@server56, 1)] with {

  ... do yet more statements (block 3) ...
```

} .

The try statement provides priority for the guarded alternatives: in essence, each guard is tested and the alternative taken is the first that can be accepted. If none can be accepted immediately then the process blocks (i.e., waits) until there is an alternative that can be accepted. If there is more than one accepting alternative, then the earliest one is taken. A pleasing consequence of this approach is that by adding a 'default' alternative with empty acquisition condition (which can always be satisfied), we can obtain the traditional non-blocking form of conditional, allowing us to express immediate tests upon resource availability.

A simple-minded translation of try from LD2k into process sum in **LSCRP** will not work because of this need to encode priority. To do this, we need to incorporate 'weighting' into the underlying process calculus — this work was done for D2k by Tofts in the context of his WSCCS [48].

4.3.6 An approach to resolving these challenges

A promising approach to attacking these issues is to proceed on the following two fronts:

1. Subsume time as a particular kind of resource in **LSCRP**, thus introducing time as a "cost" (with the obvious monoid structure);

2. Assuming an explicit granularity of atomic action — thus, hold statements would consume time as integral multiples of some minimal time (e.g., 10^{-9}). This would then allow LD2k processes to have timed synchrony, since each process would naturally decompose into sequences of atomic actions.

   Note that a side-effect is a pleasing simplification of the semantics of sequencing — it suffices to use action prefix.

5 Secure Boats: A Paradigmatic Example

5.1 Introduction

The example of this section is a variant of an important example that we call *secure boats*. This appears in in [14] together with its output trace. An earlier, more restricted version appeared in [12]. This is itself a variant of the standard *boats* example of D2k. We give both models in both the process calculus **LSCRP** and the modelling language LD2k. The variant here enables us to illustrate the use of all of the basic actions described in Section 4.

The situation we wish to model consists of a marine port with a high-security zone and a low-security zone. Submarines and boats arrive at the port and dock in the appropriate zone with the help of a cast of supporting resources. In addition, there is an information resource that may flow around the system. We intend to model a situation in which secure information may not leak, because resource transfers from the high-security area to the low-security area are rigidly controlled.

Both the LD2k and **LSCRP** models that follow have the salient features listed below.

1. There are locations: $openSea, harbour, ferryDock, subDock, debZone$. The totality of connections is given by the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ shown in Figure 5.

2. The following kinds of resource are present in the model: $ferry$, sub, $crane$, $info$, $jetty$, tug, $sectug$.

3. Ferries and submarines are resources rather than processes. It is more natural to represent such model elements as resources, since we wish to gather location information about them. The standard dual representability of D2k components as either resources or processes is less natural in the LD2k setting.

$$\begin{tikzcd}
& openSea \arrow[d, leftrightarrow] & \\
& harbour \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
ferryDock \arrow[rr] & & subDock \arrow[dl] \\
& debZone \arrow[ul] &
\end{tikzcd}$$

Figure 5: Locations for Secure Boats

4. Submarines and ferries begin at *openSea*. Submarines attempt to dock at *subDock*; ferries attempt to dock at *ferryDock*.

5. The unloading processes for both ferries and subs both use cranes. There is a potential flow of crane resources directly from the unsecure *ferryDock* to the secure location *subDock*, but not vice-versa. The link from the *ferryDock* to the *subDock* is one-way and only open some of the time (this represents a gate).

6. The docking process for subs requires the use of the information resource by the cranes.

7. There is a potential flow of crane resource from the *subDock* to the *ferryDock*, but only via the debriefing area *debZone*. The secure information is scrubbed from cranes at this point. The one-way cycle $ferryDock \rightarrow subDock \rightarrow debZone \rightarrow ferryDock$ is used to allow cranes to move between locations, whilst controlling the flow of information.

8. Both the ferryDock and subDock have two-way connections to a third area, the *harbour*. This allows vehicles (ferries and subs as resources) to move in and out of the docks to the harbour. The docking takes place by commandeering tug and jetty resources. The tugs are divided into two sorts, (ordinary) tugs and secure tugs. Ordinary tugs may only visit the ferrydock and secure tugs only the subDock. This prevents tugs from moving from subDock to ferryDock via the two-way connections to the harbour. This is desirable since tugs may transmit the information resource, but secure tugs do not.

5.2 An LD2k Executable Model

Our first model of the situation is presented in the executable code of (a version of) LD2k. The model has been executed — the resulting execution trace (too long to be included in this paper) is presented in [14]. There are some very minor differences in the syntax of the code presented in there (the model is executed as an OCAML term [32, 42]) but for our purposes this distinction is superficial.

```
(* Example LD2k model: Secure information flow control *)

(* constant declarations: no distributions yet *)
```

```
const ferryin = 2;
const ferrydelay = 3;

const ferrydocking = 5;
const feryloading = 2;
const tugreturn = 1;
const ferryreturn = 4;
const ferryout = 2;

const subdelay = 3;
const subin = 1;

const subdocking = 6;
const subloading = 2;
const stugreturn = 1;
const subreturn = 3;
const subout = 1;

const gateclosed = 12;
const gateopen = 12;

const promtime  = 1;

const scantime = 3;
const scrubtime = 2;
const cranedelay = 24;
const cranereturn = 2;

(* atomic location declarations *)
location openSea, harbour, ferryDock, subDock, debZone;

(* atomic link declarations *)
connect openSea <-> harbour;
connect harbour <-> subDock;
connect harbour <-> ferryDock;
connect ferryDock -> subDock;
connect subDock -> debZone;
connect debZone -> ferryDock;

(* initial resource distribution *)

newR(ferry@openSea,2);                          // 2 ferries at openSea
newR(sub@openSea,3);                            // 3    subs at openSea

newR(jetty@ferryDock,1);                       // 1  jetty at ferryDock
newR(crane@ferryDock,2);                       // 2 cranes at ferryDock

newR(tug@harbour,2);                          // 2       tugs at harbour
newR(sectug@harbour,4);                       // 4 sec. tugs at harbour

newR(jetty@subDock,2);                       // 2    jetties at subDock
newR(crane@subDock,1);                       // 1      crane at subDock
newR(info@subDock,5);                        // 5 info units at subDock

(* Process declarations *)

process genFerries = {
  req [ getR(ferry@openSea, 1) ];   // try to get a ferry from openSea
  moveR(ferry,openSea -> harbour ,1);  // move from openSea to harbour
  hold(ferryin);                                    // time to do move
  putR(ferry@harbour, 1);               // release ferry into harbour
  launch(genFerries, ferrydelay)                  // repeat after delay
}

process genSubs = {
  req [ getR(sub@openSea, 1) ];             // request sub from openSea
  moveR(sub,openSea -> harbour ,1);               // move sub to harbour
  hold(subin);                                          // time for move
  putR(sub@harbour, 1);                      // release sub in harbour
  launch(genSubs, subdelay)                       // repeat after delay
}

process dockFerry = {
  req[ getR(ferry@harbour, 1); getR(tug@harbour,2) ]; // request ferry
                                                          // and 2 tugs

  moveR(ferry, harbour -> ferryDock ,1);          // move ferry to dock
  try[ getR(info,harbour,1) ]          // fork on availability of info
  {
   moveR(tug , harbour -> ferryDock, 2);         //   move tugs to dock
   moveR(info, harbour -> ferryDock, 1);         // move info with tugs
   getR(jetty@ferryDock, 1);                     //          grab jetty
   hold(ferrydocking);                           //  time to dock ferry
   putR(info, harbour, 1);                   // release info at harbour
  }
  etry[] then                                       // no info grabbed
  {
   moveR(tug , harbour -> ferryDock, 2);         //   move tugs to dock
   getR(jetty@ferryDock, 1);                     //          grab jetty
   hold(ferrydocking);                           //  time to dock ferry
  };
  moveR(tug, ferryDock -> harbour, 2);       // return tugs to harbour
  putR(tug@harbour, 2);                                 // release tugs

  getR(crane@ferryDock,1);                                 // grab crane
  hold(ferryloading);                             // time to load ferry
  putR(crane@ferryDock,1);                              // release crane

  getR(tug@harbour,2);                                       // get tugs
  try[ getR(info@harbour,1) ]          // fork on ability to grab info
  {
   moveR(tug , harbour -> ferryDock, 2);      // move tugs to ferrydock
   moveR(info, harbour -> ferryDock, 1);      //    move info with tugs
   hold(tugreturn);            // time for tugs to return to ferryDock
   putR(info@ferryDock, 1);                             //   release info
  }
  etry[] then
  {
   moveR(tug, harbour -> ferryDock, 2);       // move tugs to ferryDock
   hold(tugreturn);            // time for tugs to return to ferryDock
  };
  putR(jetty@ferrydock, 1);                             // release jetty
  moveR(ferry, ferryDock -> harbour, 1); // move ferry back to harbour
  moveR(tug, ferryDock -> harbour, 2);          // move tugs to harbour
  hold(ferryreturn);                 // time to move ferry to harbour
  putR(tug@harbour,2);                              // release the tugs
  moveR(ferry, harbour -> openSea, 1);      // return ferry to open sea
  hold(ferryout);                                  // time for this return

  putR(ferry@openSea,1);                                   // release ferry

  launch(dockFerry,0)                                          // repeat
}

process dockSub = {
  req[ getR(sub@harbour, 1), getR(sectug@harbour,2) ];      // grab sub
                                                        // and sec. tugs

  moveR(sub, harbour -> subDock, 1);                 // move sub to dock
  moveR(sectug, harbour -> subDock, 2);       // move sec. tugs to dock

  getR(jetty@subDock, 1);                                   // grab jetty
  hold(subdocking);                                  // time to dock sub
  moveR(sectug, subDock -> harbour,2);  // return sec. tugs to harbour
  putR(sectug@subDock,2);                           // release sec. tugs

  try [ getR(crane@subDock,1) ] then               // try to grab crane
  {
  }
  etry [ ] then {
   req[ getR(crane@ferryDock, 1) ]      // or grab crane from ferryDock
   moveR(crane, ferryDock -> subDock, 1);      // move crane to subDock
   hold(promtime);                             //  time to do promotion
  };

  getR(info@subDock,1);             // crane needs info for loading sub
  hold(subloading);                                  // time to load sub
  putR(info@subDock, 1);             // release info on behalf of crane
  putR(crane@subDock, 1);                                // release crane

  getR(sectug@harbour,2);                                // get sec. tugs
  moveR(sectug, harbour -> subDock, 2);     // move s. tugs to subdock
  hold(stugreturn);             // time for sec. tugs to return to dock
  putR(jetty@subDock, 1);                                // release jetty
  moveR(sub, subDock -> harbour, 1);        // move sub back to harbour
  moveR(sectug, subDock -> harbour, 2);    // move sec. tugs to harbour
  hold(subreturn);                       // time to move sub to harbour
  putR(sectug@harbour,2);                       // release the sec. tugs

  moveR(sub, harbour -> openSea, 1);            // move sub to open sea
  hold(subout);                                     // time for this move
  putR(sub@openSea,1);                                     // release sub

  launch(dockSub,0);                                   // now do it again
}

process craneDeclass = {

   req[ getR(crane@subDock,1)];  // get any spare cranes from subDock
   try[ getR(info@subDock,1) ]   //                         pick up info
   {
     moveR(crane, subDock -> debZone, 1);  // move to debriefing zone
     moveR(info , subDock -> debZone, 1);  //     move info with crane
     hold(scantime);                       //    time to scan for info
```

```
      hold(scrubtime);                          //     time to scrub crane
      moveR(info, debZone -> subDock, 1);       //  return info to subDock
      putR(info@subDock,1);                     //            release info
    }
    etry[] then
    {
     hold(scantime);                                       // time to scan
    };
    moveR(crane, debZone -> ferryDock , 1); // move crane to ferryDock
    hold(cranereturn);                      //    time to return crane
    putR(crane@ferryDock,1);                //           release crane

  launch(craneDeclass, cranedelay);         //   wait before repeating
}

process gateKeeper = {

  forget(ferryDock,subDock);              //      close gate (initially)
  hold(gateclosed);                       //         time gate is closed
  recall(ferryDock,subDock);              //                   open gate

  launch(gateKeeper,gateopen);           // repeat after gateopen units
}

(* initial process launches *)
launch(genFerries, 0);
launch(genSubs, 0);
launch(gateKeeper,0);
launch(craneDeclass, 0);

do 2 {launch(dockFerry, 0); launch(dockSub, 0);} //  always have 2x2
                                                 // docking processes

hold(100);                                       // simulation duration
close;                                           //      terminate sim.
```

5.3 A Process Model in LSCRP

We now give a process algebraic model of the situation. This is slightly different model from the LD2k model above. The reasons for the differences are discussed in Subsection 5.4 below.

We use the notation '.' for the encoding of asynchronous prefix in **LSCRP**. That is, we define

$$a.E = a : \delta(E)$$

where

$$\delta(E) = E + 1 : \delta(E)$$

for all a and E, defines the *delay operator* δ. We use this to represent arbitrary integer (non-zero) time sections in the model, all of which we leave indeterminate.

An *atomic location* is a vertex of the graph $\mathcal{G}$ above. A *location (environment)* is a finite subgraph of $\mathcal{G}$. The location structure $\mathbf{L}$ is then as described in the Basic Model above. The elements are locations, ranged over by L, M, N.

An *atomic resource* is a triple (l, r, n), where l is an atomic location, r is a resource name ($ferry$, *sub*, *crane*, $info$, *jetty*, *tug* or *sectug*), and n is a natural number.

A *resource* R is a finite set of such atomic resources, such that there is a at most one n with $(l, r, n) \in R$ for each pair (l, r). Let $\mathbf{R}$ be the set of all resources. Each resource R may be viewed as a function $R : \mathcal{V} \times \mathbf{R} \longrightarrow \mathbb{N}$ as in the Basic Model. The composition of resource is given by

$$(R \circ S)(l, r) = R(l, r) + S(l, r)$$

with unit as the map that is identically zero.

The actions and processes of the process calculus are those from the Basic Model, using the location and resource parameters described above.

The model is given by a location-resource-process triple

$$L_0, R_0, E_0$$

which we shall now describe. The initial location L_0 is $\mathcal{G} - L_{(ferryDock, subDock)}$, the whole graph except for the link between the two docks. The inital resource R_0 is

$$\{(\mathit{openSea}, ferry, 2), (\mathit{openSea}, \mathit{sub}, 3), (ferryDock, jetty, 1), (ferryDock, \mathit{crane}, 2), \\ (\mathit{harbour}, \mathit{tug}, 2), (\mathit{harbour}, \mathit{sectug}, 4), (\mathit{subDock}, jetty, 2), (\mathit{subDock}, \mathit{crane}, 1), \\ (\mathit{subDock}, info, 5)\} \ .$$

The initial process E_0 is a product

$$GF \times GS \times CD \times GK \times DF \times DS$$

of processes, for which we allow the above two letter names and which we shall now describe. The process GF generates ferries (seen from the perspective of the harbour). It does this by moving ferries from the open sea into the harbour. Similarly, the process GS generates submarines. We take

$$\begin{aligned} GF &= 1 : GF + \mathsf{move}(ferry, \mathit{openSea}, \mathit{harbour}, 1) : GF \\ GS &= 1 : GS + \mathsf{move}(\mathit{sub}, \mathit{openSea}, \mathit{harbour}, 1) : GS \ . \end{aligned}$$

The gatekeeper process periodically closes and opens the link from the ferry dock to the sub dock:

$$\begin{aligned} GK = 1 : GK &+ \mathsf{forget}(ferryDock, \mathit{subDock}) : GK \\ &+ \mathsf{recall}(ferryDock, \mathit{subDock}) : GK \ . \end{aligned}$$

There is a processes to declassify cranes:

$$\begin{array}{rcllll} CD &=& 1 : CD &+& (\mathsf{move}(\mathit{crane}, \mathit{subDock}, \mathit{debZone}, 1) \\ &&&& \mathsf{move}(info, \mathit{subDock}, \mathit{debZone}, 1)) : CD_1 \\ &&&+& \mathsf{move}(\mathit{crane}, \mathit{subDock}, \mathit{debZone}, 1) : CD_2 \\ CD_1 &=& 1 : CD_1 &+& (\mathsf{move}(\mathit{crane}, \mathit{debZone}, ferryDock, 1) \\ &&&& \mathsf{move}(info, \mathit{debZone}, \mathit{subDock}, 1)) : CD \\ CD_2 &=& 1 : CD_2 &+& \mathsf{move}(\mathit{crane}, \mathit{debZone}, ferryDock, 1) : CD \ . \end{array}$$

There is a process DF to dock a ferry:

$$DF \ = \ 1 : DF + DF_1 + DF_2$$

consisting of a process to dock and unload and possibly leak information into the harbour:

$$\begin{aligned} DF_1 = \ & (\mathsf{move}(ferry, \mathit{harbour}, ferryDock, 1) \\ & \mathsf{move}(\mathit{tug}, \mathit{harbour}, ferryDock, 2) \\ & \mathsf{move}(info, \mathit{harbour}, ferryDock, 1)) : \\ & (\mathsf{get}(ferry, ferryDock, 1) \\ & \mathsf{get}(\mathit{tug}, ferryDock, 2)) : \\ & \mathsf{get}(jetty, ferryDock, 1). \\ & \mathsf{put}(\mathit{tug}, ferryDock, 2) : \\ & \mathsf{move}(\mathit{tug}, ferryDock, \mathit{harbour}, 2). \\ & \mathsf{get}(\mathit{crane}, ferryDock, 1). \\ & \mathsf{put}(\mathit{crane}, ferryDock, 1) : \\ & (DF_{11} + DF_{12}) \end{aligned}$$

or to dock and unload without leaking:

$$\begin{aligned} DF_2 = \quad & (\mathsf{move}(ferry, harbour, ferryDock, 1) \\ & \mathsf{move}(tug, harbour, ferryDock, 2)) : \\ & (\mathsf{get}(ferry, harbour, ferryDock, 1) \\ & \mathsf{get}(tug, harbour, ferryDock, 2)) : \\ & \mathsf{get}(jetty, ferryDock, 1) : \\ & \mathsf{put}(tug, ferryDock, 2) : \\ & \mathsf{move}(tug, ferryDock, harbour, 2). \\ & \mathsf{get}(crane, ferryDock, 1). \\ & \mathsf{put}(crane, ferryDock, 1) : \\ & (DF_{11} + DF_{12}) \end{aligned}$$

with sub-processes to 'undock' with leakage:

$$\begin{aligned} DF_{11} = \quad & (\mathsf{move}(info, harbour, ferryDock, 1) \\ & \mathsf{move}(tug, harbour, ferryDock, 2)) : \\ & \mathsf{get}(tug, ferryDock, 1)). \\ & \mathsf{put}(jetty, ferryDock, 1) : \\ & \mathsf{put}(tug, ferryDock, 2) : \\ & (\mathsf{move}(ferry, ferryDock, harbour, 1) \\ & \mathsf{move}(tug, ferryDock, harbour, 2)). \\ & \mathsf{move}(ferry, harbour, openSea, 1). \\ & DF \end{aligned}$$

or undock without leakage:

$$\begin{aligned} DF_{12} = \quad & \mathsf{move}(tug, harbour, ferryDock, 2). \\ & \mathsf{put}(jetty, ferryDock, 1) : \\ & (\mathsf{move}(ferry, ferryDock, harbour, 1) \\ & \mathsf{move}(tug, ferryDock, harbour, 2)). \\ & \mathsf{move}(ferry, harbour, openSea, 1). \\ & DF \ . \end{aligned}$$

Similarly, there is a process to dock a sub:

$$DS \quad = \quad 1 : DS + DS_1$$

consisting of a process to grab the sub and secure tugs and move them from the harbour to the subDock:

$$\begin{aligned} DS_1 = \quad & (\mathsf{move}(sub, harbour, subDock, 1) \\ & \mathsf{move}(sectug, harbour, subDock, 2)) : \\ & (\mathsf{get}(sub, subDock, 1) \\ & \mathsf{get}(sectug, subDock, 2)) : \\ & \mathsf{get}(jetty, subDock, 1). \\ & \mathsf{put}(sectug, subDock, 2) : \\ & \mathsf{move}(sectug, subDock, harbour, 2). \\ & (DS_{11} + DS_{12}) \ . \end{aligned}$$

A non-deterministic choice is then made to grab an available crane already in the subDock:

$$DS_{11} = \quad \mathsf{get}(crane, subDock, 1).DS_2$$

or we move a crane to the subDock, consuming time asynchronously in so doing:

$$\begin{aligned} DS_{12} = \quad & \mathsf{move}(crane, ferryDock, subDock, 1). \\ & \mathsf{get}(crane, ferryDock, 1). \\ & DS_2 \end{aligned}$$

We complete each choice in DS with the process DS_2 and finally continue with DS:

$$\begin{aligned} DS_2 = \;& \mathsf{get}(info, subDock, 1). \\ & \mathsf{put}(info, subDock, 1) : \\ & \mathsf{put}(crane, subDock, 1) : \\ & \mathsf{move}(sectug, harbour, subDock, 2). \\ & \mathsf{put}(jetty, subDock, 1) : \\ & \mathsf{put}(sub, subDock, 1) : \\ & (\mathsf{move}(sub, subDock, harbour, 1) \\ & \mathsf{move}(sectug, subDock, harbour, 2)). \\ & \mathsf{move}(sub, harbour, openSea, 1). \\ & DS\ . \end{aligned}$$

5.4 Comparison of the Models

We reiterate that our two models are not identical, and that this is entirely deliberate. Slightly different aspects of the same situation are *easy* to capture with the two different methodologies. However, this does *not* mean that the two methodologies express fundamentally different concepts of location, resource and process.

In particular, the process calculus model above captures the intent of the LD2k model only partially. The following aspects of LD2k are in addition to the points discussed in §4.3:

1. Timing concerns are of central importance in the LD2k model;
2. Certain actions which are synchronized in the process model are sequenced in the LD2k model (e.g. moving the ferry then the tugs) but with no timing difference between the actions. This difference can impact upon resource use and so affect the possible traces of the models;
3. In the LD2k model, moves are performed upon resources which are owned by the process which does the moving. In the process calculus this is not the case. Thus an instance of $\mathsf{move}(r, l, l', n)$ in the process model corresponds broadly to an LD2k sequence:

   ```
   getR(r@l, n) ; moveR(r, l -> l', n) ;  putR(r@l',n)
   ```

   with no hold(t) instances between the getR and the putR;
4. Our process model here does not capture the subtleties of priority intended by a try statement in LD2K/D2K models. Instead we merely use non-deterministic choice to represent each alternative possibility. Because we can only test for the presence of each resource by claiming it, this means that we cannot eliminate the potential for overlapping access and hence prevent the possibility of claiming the 'wrong' resource. In short, the try statement in both LD2k and Demos2k is a form of prioritized choice. A weighted refinement of **LSCRP**, along the lines of [48], would allow us to express the full detail of these aspects of the tool.

6 Logic

6.1 Syntax

We now introduce a logic of **OLSCRP**-system properties. The logic is a semantically defined system in the style of Hennessy-Milner logic [23], based on the bunched logic BI. In addition to BI's basic (intuitionistic) additives and multiplicatives, the logic of **OLSCRP**-system properties admits both additive and multiplicative quantifiers and both additive and multiplicative modalities.

Some features of the Hennessy-Milner style forcing relation are noteworthy:

- Multiplicative conjunction characterizes the (synchronous) concurrent composition;

- Multiplicative quantification characterizes hiding;
- The multiplicative modalities express properties of processes in the presence of additional resources at extended locations.

Assume a collection of action variables x, y that range over the set of actions, Act. Assume a collection of atomic formulae, ranged over by p. Define the set of atomic propositions by

$$\begin{array}{rl}\phi ::= & p \mid \bot \mid \phi \vee \psi \mid \top \mid \phi \wedge \psi \mid \phi \rightarrow \psi \mid \langle \mathsf{a} \rangle \phi \mid [\mathsf{a}]\phi \mid \exists x.\phi \mid \forall x.\phi \\ & \mid I \mid \phi * \psi \mid \phi -\!\!* \psi \mid \langle \mathsf{a} \rangle_\nu \phi \mid [\mathsf{a}]_\nu \phi \mid \exists_\nu x.\phi \mid \forall_\nu x.\phi\end{array}$$

where a is either an action a or an action variable x. Let ϕ, ψ range over such propositions. The conectives $*$, $-\!\!*$, I, $\langle \mathsf{a} \rangle_\nu$, $[\mathsf{a}]_\nu$, $\exists_\nu$, $\forall_\nu$ are the *multiplicative conjunction, implication, unit, modalities and quantifiers*. We refer to this language as **LMBIi**.

6.2 Interpretation

The logical language above has an intepretation on the states of **OLSCRP**. First, we introduce a few more notions before we give the interpretation.

We make use of the order on states given by

$$L, R, E \sqsubseteq M, S, F \quad \text{iff} \quad L \preceq M \text{ and } R \sqsubseteq S \text{ and } E \sim F$$

for all L, M, R, S, E, F.

We use a binary relation, $\circledless$, on resources given by: $S \circledless R$ iff for any T, the composite $S \circ T$ is defined whenever $R \circ T$ is defined. This relation holds for all pairs of resources when the resource monoid is total. This is the case for LD2k resources, as explained above.

An *(action) environment* is a partial function, η from action variables to actions. Environments are partially ordered by inclusion of their graphs: write $\eta \sqsubseteq \eta'$ iff $\eta(x) = \eta'(x)$ for all action variables x such that $\eta(x)$ is defined. Say that η and η' are *consistent* if they agree on all x such that both are defined. Write $\eta \circ \eta'$ for the environment formed by taking the union of the graphs of η and η', and note that this is well-defined just when η and η' are consistent.

In what follows, it is convenient to extend all environments with the identity map on actions, so that they map all actions (in Act) to themselves: we do not distinguish the two notions with any particular notation and refer to both notions as environments.

The semantics that will be presented below is a variant of a Kripke model for first-order logic. Thus the state space is presented at stages-of-definition, indexed so that the state space develops as variables are assigned to constants. Define the *state space at* η to be the transition system with the same states as **OLSCRP** and transitions

$$L, R, E \xrightarrow{\mathsf{a}}_\eta L', R', E'$$

whenever $\eta(\mathsf{a}) \downarrow$ and $L, R, E \overset{\eta(\mathsf{a})}{\rightarrow} L', R', E'$.

A *valuation* is a map from atomic formulae to the set of functions from environments to states such that, if the free variables of p are $x_1, \ldots, x_n$ then the set of states $\mathcal{V}(p)\eta$ depends only upon p and the values $\eta(x_1), \ldots, \eta(x_n)$, and is the same for any other η' that agrees with η on these variables.

We further assume that all such valuations are closed under the following:

- if $L, R, E \in \mathcal{V}(p)\eta$ and $L \preceq M$ then $M, R, E \in \mathcal{V}(p)\eta$
- if $L, R, E \in \mathcal{V}(p)\eta$ and $R \sqsubseteq S$ then $L, S, E \in \mathcal{V}(p)\eta$
- if $L, R, E \in \mathcal{V}(p)\eta$ and $E \sim F$ then $L, R, F \in \mathcal{V}(p)\eta$.

The valuation is extended to an interpretation of all formulae using the forcing relation of Figures 6, 7.

$$L,R,E,\eta \vDash p(x_1,\dots x_n) \text{ iff } L,R,E \in \mathcal{V}(p)\eta$$

$$L,R,E,\eta \vDash \bot \text{ never} \qquad L,R,E,\eta \vDash \top \text{ always}$$

$$L,R,E,\eta \vDash \phi \wedge \psi \text{ iff } L,R,E,\eta \vDash \phi \text{ and } L,R,E,\eta \vDash \psi$$

$$L,R,E,\eta \vDash \phi \vee \psi \text{ iff } L,R,E,\eta \vDash \phi \text{ or } L,R,E,\eta \vDash \psi$$

$$L,R,E,\eta \vDash \phi \to \psi \text{ iff } \forall M,S,\eta'.\ L \preceq M \text{ and } R \sqsubseteq S \text{ and } \eta \sqsubseteq \eta' \text{ and } M,S,E,\eta' \vDash \phi \text{ implies } M,S,E,\eta' \vDash \psi$$

$$L,R,E,\eta \vDash \langle \mathsf{a}\rangle\phi \text{ iff } \exists E'.\ L,R,E \overset{\eta(\mathsf{a})}{\to} \mu(\eta(\mathsf{a}),L,R),E' \text{ and } \mu(\eta(\mathsf{a}),L,R),E',\eta \vDash \phi$$

$$L,R,E,\eta \vDash [\mathsf{a}]\phi \text{ iff } \forall M,S,E'.\ L \preceq M \text{ and } R \sqsubseteq S \text{ and } \eta \sqsubseteq \eta' \quad M,S,E \overset{\eta'(\mathsf{a})}{\to} \mu(\eta'(\mathsf{a}),M,S),E' \text{ implies } \mu(\eta'(\mathsf{a}),M,S),E',\eta' \vDash \phi$$

$$L,R,E,\eta \vDash \exists x.\phi \text{ iff } (\exists x.\ \eta(x)=a) \text{ and } L,R,E,\eta \vDash \phi[a/x]$$

$$L,R,E,\eta \vDash \forall x.\phi \text{ iff } \forall \eta'.\forall a.\ \eta \sqsubseteq \eta' \text{ and } (\exists x.\eta'(x)=a) \text{ implies } L,R,E,\eta' \vDash \phi[a/x]$$

Figure 6: Interpretation of atomic and additive formulae

$$L,R,E,\eta \vDash I \text{ iff } e \sqsubseteq R \text{ and } E \sim 1$$

$$L,R,E,\eta \vDash \phi_1 * \phi_2 \text{ iff } \exists R_1,R_2,E_1,E_2.\ R_1 \circ R_2 \sqsubseteq R \text{ and } E_1 \times E_2 \sim E \text{ and } L,R_1,E_1,\eta \vDash \phi_1 \text{ and } L,R_2,E_2,\eta \vDash \phi_2$$

$$L,R,E,\eta \vDash \phi \mathrel{-\!\!*} \psi \text{ iff } \forall M,S,T,F,\eta'.\ R \circ S \sqsubseteq T \text{ and } L \preceq M \text{ and } \eta \circ \eta' \downarrow \text{ and } M,S,F,\eta' \vDash \phi \text{ implies } M,T,E \times F,\eta \circ \eta' \vDash \psi$$

$$L,R,E,\eta \vDash \langle \mathsf{a}\rangle_\nu\phi \text{ iff } \exists S.\ R \circ S \downarrow \text{ and } \forall M,T.\ L \preceq M \text{ and } R \circ S \sqsubseteq T \text{ implies } \exists E'.\ M,T,E \overset{\eta(\mathsf{a})}{\to} \mu(\eta(\mathsf{a}),M,T),E' \text{ with } \mu(\eta(\mathsf{a}),M,T),E',\eta \vDash \phi$$

$$L,R,E,\eta \vDash [\mathsf{a}]_\nu\phi \text{ iff } \forall M,S,T,E',\eta'.\ L \preceq M \text{ and } R \circ S \downarrow \text{ and } R \circ S \sqsubseteq T \text{ and } \eta \sqsubseteq \eta' \text{ and } M,T,E \overset{\eta'(\mathsf{a})}{\to} \mu(\eta'(\mathsf{a}),M,T),E' \text{ implies } \mu(\eta'(\mathsf{a}),M,T),E',\eta' \vDash \phi$$

$$L,R,E,\eta \vDash \exists_\nu x.\phi \text{ iff } \exists S,F,a,x.\ \eta(x)=a \text{ and } R \circ S\downarrow \text{ and } \mu(a,L,S)\downarrow \text{ and } S \otimes R \text{ and } E \sim \nu S.F \text{ and } L,R \circ S,F,\eta \vDash \phi[a/x]$$

$$L,R,E,\eta \vDash \forall_\nu x.\phi \text{ iff } \forall M,S,F,a,\eta'.\ L \preceq M \text{ and } R \circ S\downarrow \text{ and } \eta \sqsubseteq \eta' \text{ and } \mu(a,M,R \circ S)\downarrow \text{ and } E \sim \nu S.F \text{ implies } M,R \circ S,F,\eta' \vDash \phi[a/x]$$

Figure 7: Interpretation of multiplicative formulae

6.3 Theory

The theorem below (which is half of the appropriate Hennessy-Milner theorem in this setting) holds.

Theorem 1. If $L, R, E, \eta \vDash \phi$ and $E \sim F$ then $L, R, F, \eta \vDash \phi$.

Proof. The proof is the by induction on the structure of ϕ. We omit the details as they are similar to those found in [16]. However, the additive modalities cases work because the relation $\sim$ on states is contained in the relation $\approx$. The case $-\!\!*$ works because $\sim$ is a congruence. The multiplicative modalities require both of these facts. $\square$

Also, we have the following monotonicity property, again proved by the evident induction on the structure of ϕ. In particular, the $\circledcirc$ predicate is used in the proof.

Theorem 2. If $L, R, E, \eta \vDash \phi$ and $L, R, E \sqsubseteq M, S, F$ and $\eta \sqsubseteq \eta'$ then $M, S, F, \eta' \vDash \phi$ holds.

If the partial order on locations and resources is taken to be discrete, and we restrict our attention to total environments then the interpretation of connectives is, essentially, that which one would expect of a classical logic. Negation (interpretated by complementation) agrees with the interpretation of the pseudocomplement $((-)\to\bot)$.

In this classical setting [16], for the fragment of **LMBIi** without

$-\!\!*$, $\langle a\rangle_\nu$, and $[a]_\nu$

an interpretation can be given that uses the local bisimulation relation $\approx$, such that this interpretation satisfies Theorem 1 (modifying [16]). In this situation, the standard proof [23, 40, 41, 45] of the converse to Theorem 1 can be easily adapted (assuming a classical meta-theory). Note that congruence for $\approx$ is not required for this proof.

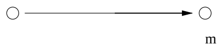
Theorem 3. If, forall ϕ and η, ($L, R, E, \eta \vDash \phi$ iff $L, R, F, \eta \vDash \phi$), then, for the fragment given above, $L, R, E \approx L, R, F$.

In fact, any fragment including $\top$, $\wedge$, $\neg$, and $\langle a\rangle$ will allow this theorem to go through.

6.4 Examples

Example 6. We construct an example in the process language of Section 4 above.

Suppose that there are two atomic locations l and m. Consider a simple location L_1 in which there is just one link, and this has source l and target m.



Suppose that there is just one resource name r. A resource R can then be represented by a pair (p, q) of natural numbers, where p is the quantity of resource at l and q is the quantity of resource at m. Composition is componentwise addition. Suppose that there is an initial resource $R_1 = (1, 0)$, and a process $E_1 := \mathsf{move}(r, l, m, 1) : 1 + 1 : E_1$.

An important consequence of the splitting of system state into location, resource and process is that it is possible to have simple, atomic assertions about location and resource in the logical assertion language. For any (p, q) let $\phi_{(p,q)}$ be the atomic assertion that the current resource distribution is (p, q). For any atomic locations l and m let $\psi_{(l,m)}$ be the atomic assertion that there is a link from l to n. Then,

$$L_1, R_1, E_1 \vDash \phi_{(1,0)} \wedge \psi_{(l,m)} \wedge \neg\psi_{(m,l)} \wedge \neg\psi_{(l,l)} \wedge \neg\psi_{(m,m)}$$

completely describes the initial location graph and resource distribution.

Simple modal assertions describe the one-step evolution of the system as follows:

$$L_1, R_1, E_1 \vDash \langle \mathsf{move}(r, l, m, 1)\rangle\phi_{(0,1)} \wedge [\mathsf{move}(r, l, m, 1)]\phi_{(0,1)} \wedge \langle 1\rangle\phi_{(1,0)} \wedge [1]\phi_{(1,0)} \ .$$

If there is any other process G such that G has a tick action, then

$$L_1, R_1, E_1 \times G \vDash \langle \mathsf{move}(r, l, m, 1) \rangle \phi_{(0,1)}$$

and so

$$L_1, \mathbf{e_R}, E_1 \vDash (\phi_{(1,0)} \wedge \langle 1 \rangle \top) \multimap \langle \mathsf{move}(r, l, m, 1) \rangle \phi_{(0,1)} \tag{1}$$

holds.

Much of the force of such $\multimap$-statements is that they allow for a kind of modular reasoning, in which we move from assertions about parts of systems to assertions about whole systems. For example, with $G_1 := (1 : G_1) \times (\mathsf{get}(r, m, 1) : \mathsf{0})$ we have $L_1, R_1, G_1 \vDash (\phi_{(1,0)} \wedge \langle 1 \rangle \top)$ and so for the larger system

$$L_1, R_1, E_1 \times G_1 \vDash \langle \mathsf{move}(r, l, m, 1) \rangle \phi_{(0,1)}$$

using (1) above. Of course, this particular assertion could also have been produced directly using the semantics of the additive diamond modality.

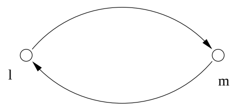
As an example of the use of a multiplicative modality we have,

$$L_1, \mathbf{e_R}, G_1 \vDash ([\mathsf{get}(r, m, 1)] \bot) \wedge \langle \mathsf{get}(r, m, 1) \rangle_\nu \phi_{(0,0)}$$

since the get action can occur just when there is at least one unit of resource at location m. $\square$

Example 7. Again, this is an example in the style of Section 4.

Suppose that there are two atomic locations l and m. Let L_2 be the location consisting of l, m, a link from l to m, and a link from m to l. Suppose that there are two resource names r and s.



Resources are now described by quadruples $(p^r_l, p^r_m, p^s_l, p^s_m)$, where each p^j_i is the quantity of resource j at location i. Let $\phi_{(p,q,i,j)}$ be the assertion that the current system has resources (p, q, i, j). We focus here on a situation in which there is at most one unit of r summed over all locations, and similarly for s. Each such unit can be regarded as a token representing the position of that resource. So, for example, if there is one unit of resource r at vertex l then the position of r is l.

Consider processes

$$\begin{array}{lcl} E_2 & := & (1 : E_2) + (\mathsf{move}(r, l, m, 1) : E_2) + (\mathsf{move}(r, m, l, 1) : E_2) \\ F_2 & := & (1 : F_2) + (\mathsf{move}(s, l, m, 1) : F_2) + (\mathsf{move}(s, m, l, 1) : F_2) \\ G_2 & := & (1 : G_2) + (\mathsf{get}(r, l, 1)\mathsf{get}(s, l, 1) : \mathsf{0}) + (\mathsf{get}(r, m, 1)\mathsf{get}(s, m, 1) : \mathsf{0}) \ . \end{array}$$

The system $L_2, (1, 0, 0, 1), E_2 \times F_2 \times G_2$ cycles resources around the graph L_2 until they lie at the same vertex at the same time. At this point the G_2 component of the system may simultaneously claim both of the resources and then terminate.

Let θ_1 be $((\phi_{(0,1,0,0)} \wedge \langle \mathsf{move}(r, m, l, 1) \rangle \top)$. Let θ_2 be $(\phi_{(0,0,1,0)} \wedge \langle \mathsf{move}(s, l, m, 1) \rangle \top)$. Let θ_3 be the assertion $\phi_{(0,0,0,0)}$.

Then

$$\begin{array}{lcl} L_2, (1, 0, 0, 1), E_2 \times F_2 \times G_2 & \vDash & (\phi_{(1,0,0,0)} \wedge \langle \mathsf{move}(r, l, m, 1) \rangle \theta_1) * \\ & & (\phi_{(0,0,0,1)} \wedge \langle \mathsf{move}(r, l, m, 1) \rangle \theta_2) * \\ & & \langle 1 \rangle \theta_3 \end{array}$$

since, in particular,

$$\begin{array}{lcl} L_2, (1, 0, 0, 0), E_2 & \vDash & \langle \mathsf{move}(r, l, m, 1) \rangle \theta_1 \\ L_2, (0, 0, 0, 1), F_2 & \vDash & \langle \mathsf{move}(s, m, l, 1) \rangle \theta_2 \\ L_2, (0, 0, 0, 0), G_2 & \vDash & \langle 1 \rangle \theta_3 \end{array}$$

all hold. We also have

$$L_2, (1, 0, 0, 1), E_2 \times F_2 \times G_2 \quad \vDash \quad \langle \mathsf{move}(r, l, m, 1)\mathsf{move}(s, m, l, 1) \rangle (\theta_1 * \theta_2 * \theta_3)$$

so that, after the positions of the r and s tokens are exchanged the first time, they may be exchanged a second-time.

More generally, for any assertions $\phi_1, \dots \phi_n$ we have

$$L, R, E \vDash \forall x_1 \dots \forall x_n((\langle x_1\rangle\phi_1) * \dots * (\langle x_n\rangle\phi_n)) \text{ implies } L, R, E \vDash \langle x_1 \dots x_n\rangle(\phi_1 * \dots * \phi_n)$$

for all L, R, E. □

Example 8. Consider the example L_0, R_0, E_0 from Section 5. This satisfies

$$L_0, R_0, E_0 \vDash \phi_1 * \phi_2$$

where ϕ_1 is 'there are two free tugs' and ϕ_2 is 'there are two free secure tugs'. The fact that the model satisfies this multiplicative conjunction in one of the facts that allows ferries and submarines to dock simultaneously using the DF and DS processes since the tugs can used by DF and the secure tugs by DS.

This can reach a state L, R, E that has no $\mathsf{get}(crane, subDock, 1)$-transition, because it lacks a free crane at the $subDock$. Such a state does not satisfy the relation

$$L, R, E \vDash \langle \mathsf{get}(crane, subDock, 1)\rangle\top$$

and so a paticular submarine cannot currently dock. There are, however, such states that do satisfy

$$L, R, E \vDash \langle \mathsf{get}(crane, subDock, 1)\rangle_\nu\top$$

so that the submarine may dock given additional resource (another crane). This suggets the structural property of models (which we have) that cranes can be drafted in to the $subDock$ along a connection from another location (in our case the $ferryDock$).

Let ϕ be the assertion 'the information component of the resource at $ferryDock$ is zero'. Then

$$L_0, R_0, E_0 \vDash \forall x_1 \dots x_n.[x_1] \dots [x_n]\phi$$

asserts the valid statement that information is not leaked in n-steps. It seems that a more powerful language with, say, infinitary conjunction, or quantification over the natural numbers is needed for making the assertion that information never leaks.

In the above example, the underlying location structure $\mathcal{G}$ is small. In larger examples, it may be useful to have a logical language with variables for atomic locations (and possibly atomic links) in atomic formulae and quantification over such variables. □

Further examples of the use of $*$ and $-\!\!*$ in reasoning about resource and process (but not location) are given in [40, 41]. More complex properties of systems often require enrichments of the logical language presented here, for example 'always' or 'eventually' modalities.

7 Discussion

We have produced a theoretical framework that encompasses both the dynamic behaviour of discrete event systems with distributed resource and a logic for formal reasoning about system properties. We believe that this leads to scalable, practical tools for systems modelling. We have described a kernel of a tool (LD2k) which demonstrates our ideas.

Nevertheless, much work remains fully to deliver a well-founded, practical tool. The relationship between **LSCRP** and LD2k is clearly very close: indeed, we claim that **LSCRP** is the correct foundation for LD2k. That said, making this correspondence formal requires a good deal of further work, and clearly involves giving a semantics to LD2k using **LSCRP**. This should take the form of a (completely formalized) translation that defines LD2k phrases and commands by the use of **LSCRP** expressions.

As part of developing our current LD2k framework further, it will be necessary to make a large number of extensions to give a better-rounded platform for developing large-scale systems models. Applied modellers expect to have all of the same kinds of language features that are familiar from using D2k — but extended with location concepts.

References

[1] J. Baeten and J. Bergstra. Real space process algebra. *Formal Aspects of Computing*, 5:481–529, 1993.

[2] J. Barwise and J. Seligman. *Information Flow: The Logic of Distributed Systems.* C.U.P, 1997.

[3] A. Beautement, R. Coles, J. Griffin, B. Monahan, D. Pym, M. A. Sasse, and M. Wonham. Modelling the human and technological costs and benefits of usb memory stick security. In M. Johnson, editor, *Managing Information Risk and the Economics of Security.* Springer, Forthcoming (5th December 2008). Conference version at: `http://weis2008.econinfosec.org/papers/Pym.pdf`.

[4] Y. Beres, J. Griffin, and S. Shiu. Security analytics: Analysis of security policies for vulnerability management. Technical Report HPL-2008-121, HP Labs, 2008. Conference version to appear as: 'Analyzing the performance of security operations to reduce vulnerability exposure windows', Y. Beres, J. Griffin, S. Shiu, M. Heitman, D. Markle, P.Ventura, Annual Computer Security Applications Conference (ACSAC) 2008, IEEE.

[5] N. Biri and D. Galmiche. Models and separation logics for resource trees. *Journal of Logic and Computation*, 4(17):687–726, 2007.

[6] G. Birtwistle. *Demos — discrete event modelling on Simula.* Macmillan, 1979.

[7] G. Birtwistle, R. Pooley, and C. Tofts. Characterising the structure of simulations using CCS. *Transactions of the Simulation Society*, 10(3):205–236, 1993.

[8] C. Calcagno, P. Gardner, and U. Zarfaty. Context logic and tree update. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages 2005 (POPL)*, pages 271–282, 2005.

[9] L. Cardelli and A. Gordon. Mobile ambients. *Theoret. Comp. Sci.*, 240:177–213, 2000.

[10] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems (CAV '89)*, volume 407 of *Lecture Notes in Computer Science*, pages 24–37. Springer-Verlag, 1989.

[11] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based verification tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, 1993.

[12] M. Collinson, B.Monahan, and D. Pym. Located Demos2k: A tool for executing processes relative to distributed resources. Technical Report HPL-2008-76, HP Labs, 2008.

[13] M. Collinson, B.Monahan, and D. Pym. A logical and computational theory of located resource. Technical Report HPL-2008-74R1, HP Labs, 2008.

[14] M. Collinson, B.Monahan, and D. Pym. An Update to Located Demos2k. Technical Report HPL-2008-205, HP Labs, 2008.

[15] M. Collinson and D. Pym. Algebra and logic for access control. Technical Report HPL-2008-75R1, HP Labs, 2008.

[16] M. Collinson and D. Pym. Algebra and logic for resource-based systems modelling. Submitted: `http://www.cs.bath.ac.uk/~pym/mbi.pdf`, 2008.

[17] M. Collinson, D. Pym, and C. Tofts. Errata for Formal Aspects of Computing (2006) 18:495–517 and their consequences. *Formal Aspects of Computing*, 19(4):551–554, 2007.

[18] G. Conforti, D. Macedonio, and V. Sassone. Static bilog: a unifying language for spatial structures. *Fundamenta Informaticae*, 80:1–20, 2007.

[19] O.-J. Dahl, B. Myhrhaug, and K. Nygaard. Simula 67 common base language. NCC Publication S-52, Norwegian Computing Center, Oslo, 1970.

[20] R. De Nicola and M. Loreti. A modal logic for mobile agents. *ACM Transactions on Computational Logic*, 5(1):79–128, 2004.

[21] Demos2k. `http://www.demos2k.org`.

[22] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 322:615–669, 2003.

[23] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[24] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[25] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. POPL 2001*, pages 14–26. ACM, 2001.

[26] O. Jensen and R. Milner. Bigraphs and mobile processes (revised). Technical Report 580, Computer Laboratory, University of Cambridge, 2004.

[27] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[28] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[29] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

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

[31] R. Milner. Bigraphs as a model for mobile interaction. In *Graph Transformation*, volume 2505 of *Lecture Notes in Computer Science*, pages 8–13. Springer Berlin / Heidelberg, 2002.

[32] OCAML. `http://caml.inria.fr/`.

[33] P. O'Hearn. Resources, concurrency and local reasoning. *Theoretical Computer Science*, 375(1–3):271–307, 2007.

[34] P. O'Hearn and D. Pym. The logic of bunched implications. *Bull. Symb. Logic*, 5(2):215–244, 1999.

[35] G. Plotkin. Structural operational semantics. *Journal of Logic and Algebraic Programming*, 60:17–139, 2004. Original manuscript 1981.

[36] S. Popkorn. *First Steps in Modal Logic*. Cambridge University Press, 1994.

[37] D. Pym. On bunched predicate logic. In *Proc. LICS'99*, pages 183–192. IEEE, 1999.

[38] D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002. Errata at: `http://www.cs.bath.ac.uk/~pym/BI-monograph-errata.pdf`.

[39] D. Pym, P. O'Hearn, and H. Yang. Possible worlds and resources: The semantics of **BI**. *Theoretical Computer Science*, 315(1):257–305, 2004.

[40] D. Pym and C. Tofts. A calculus and logic of resources and processes. *Formal Aspects of Computing*, 18(4):495–517, 2006. Errata in [17].

[41] D. Pym and C. Tofts. Systems Modelling via Resources and Processes: Philosphy, Calculus, Semantics, and Logic. In L. Cardelli, M. Fiore, and G. Winskel, editors, *Computation, Meaninig and Logic: articles dedicated to Gordon Plotkin*, volume 107 of *Electronic Notes in Theoretical Computer Science*, pages 545–587. Elsevier, 2007. Errata in [17].

[42] D. Rémy. Using, Understanding, and Unraveling the OCaml Language. In G. Barthe, editor, *Applied Semantics. Advanced Lectures*, number 2395 in LNCS, pages 413–537. Springer Verlag, 2002.

[43] J. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proc. LICS'02*, pages 55–74. IEEE, 2002.

[44] J. Riely and M. Hennessy. Distributed processes and location failures. *Theoretical Computer Science*, 226:693–735, 2001.

[45] C. Stirling. *Modal and temporal properties of processes*. Springer, 2001.

[46] R. Taylor, C. Tofts, and M. Yearworth. Open Analytics. Technical Report HPL-2004-138R1, HP Labs, 2004.

[47] The Concurrency Workbench. `http://www.lfcs.inf.ed.ac.uk/cwb/`.

[48] C. Tofts. Processes with probabilities, priority and time. *Formal Aspects of Computing*, 6:536–564, 1994.

[49] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In D. Dill, editor, *CAV'94: Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.

[50] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. Technical Report DoCS 94/45, Department of Computer Systems, Uppsala University, Sweden, February 1994. Also available as Technical Report ECS-LFCS-94-285, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK.

[51] M. Yearworth, B. Monahan, and D.Pym. Predictive modelling for security operations economics (extended abstract). Technical Report HPL-2006-125, HP Labs, 2006. Also appeared at: Proc. I3P Workshop on the Economics of Securing the Information Infrastructure (WESSI), 2006, `http://wesii.econinfosec.org/workshop/`.