CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 225 (2009) 3–19
www.elsevier.com/locate/entcs

Sheaves, Objects, and Distributed Systems

Grant Malcolm[1]

Department of Computer Science
University of Liverpool
Liverpool, UK

Abstract

We review and extend some recent work that uses sheaf theory to provide a semantic foundation for distributed concurrent systems. A sheaf can be thought of as a system of observations on a topological space, with the key property that consistent local observations can be uniquely pasted together to provide a global observation. We suggest that sheaf theory can provide a framework for the semantics of distributed concurrent systems by exploring the relationships between sheaves and basic models of concurrent processes, particularly labelled transition systems and algebraic specifications of classes and objects.

Keywords: Distributed systems, sheaf theory, algebraic specification, concurrency.

1 Introduction

In this paper we explore the possibility of using sheaf theory to provide a semantic foundation for distributed concurrent systems. We are particularly interested in systems of interacting objects with hierarchical structure: objects can be built by aggregation from other objects, and communication takes place through shared subobjects. We aim to give a fairly non-technical presentation: rather than give definitions and theorems, we give only a few definitions, and illustrate constructions through examples. Some category theory is occasionally used, but we try to keep the discussion at an intuitive level.

Sheaf theory is concerned with the transition from local to global properties. A sheaf can be thought of as a system of observations on a topological space, with the key property that consistent local observations can be uniquely pasted together to provide a global observation. Although these intuitions behind sheaves are clearly relevant to the study of distributed systems, their application in Computer Science has been sporadic. An early use of sheaf theory was a paper by Monteiro and Pereira [25], who applied sheaf theory to study connections between event systems. More

[1] Email: grant@csc.liv.ac.uk

1571-0661/© 2008 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2008.12.063

general papers by Ehrich, Goguen and Sernadas [5] and by Goguen [11] advocated sheaves over a linear topology (effectively linear discrete time) as a foundation for a behavioural account of concurrent processes. This approach was followed by Cîrstea [2] in providing a semantics for a concurrent object-oriented programming language. More recent work by Monteiro [24] uses 'observation systems', which are very closely related to sheaves, to study coalgebras. All of these works are very promising, and the present paper is intended to contribute to the application of sheaves to distributed concurrent systems by exploring the relationships between sheaves and basic models of concurrent processes, particularly labelled transition systems and algebraic specifications of classes and objects.

Our interest in sheaves arose from reading some early papers by Joseph Goguen [7,8,9] on what he called Categorical Systems Theory. Some slogans, or general principles, from that approach also recur in the paper [11], which explicitly relates sheaves and concurrent interacting objects. Among those that are relevant to the present work are:

- Objects are Sheaves
- Systems are Diagrams, and
- Behaviour is Limit.

Section 2 illustrates these slogans by looking at sheaves and two basic structures used in semantics: transition systems and algebraic specifications. We show that transitions systems provide examples of sheaves, a result very much in line with the body of work on presheaf models of concurrency [1]. We also give a brief account of a kind of object-aggregation construction in 'hidden algebra', to motivate the generalisation of sheaf used in Section 3. This gives a non-technical account of work in [21], that used sheaves of object specifications. We relate the limit constructions of Section 2 to this more general approach of sheaves of transition systems or sheaves of object specifications. Section 3 concludes with a more speculative approach that sketches out how sheaves of objects can be used to model dynamic systems: systems where the topology of objects and subobjects changes. This approach is inspired by some as-yet unpublished work arising in the Irish School of Constructive Mathematics.

2 Transition Systems, Sheaves and Components

This section presents some basic definitions and constructs on labelled transition systems, as an example of the kind of hierarchical approach to object systems that we are interested in. We also give an introduction to sheaves, and use transition systems as an example of sheaves. We conclude the section with another approach to hierarchical systems using hidden algebraic specifications.

2.1 Labelled Transition Systems

We begin with a standard definition of transition system, which we later generalise in order to capture hierarchically structured systems.

Definition 2.1 A **labelled transition system over** L is a pair $(T, \longrightarrow)$, where T is a set of **states**, and $\longrightarrow \subseteq T \times L \times T$ is the **transition relation**. We write $t \xrightarrow{l} t'$ for $(t, l, t') \in \longrightarrow$.

A **morphism** of transition systems over L $(T_1, \longrightarrow_1) \to (T_2, \longrightarrow_2)$ is a function $f : T_1 \to T_2$ such that if $t \xrightarrow{l}_1 t'$ then $f(t) \xrightarrow{l}_2 f(t')$.

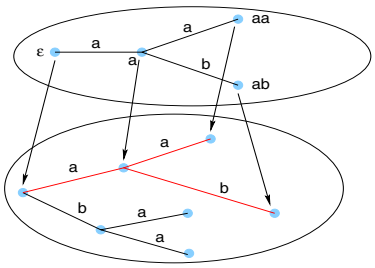
When there is no risk of confusion, we will drop subscripts and decorations on the arrow '$\longrightarrow$', and simply write, for example, 'if $t \xrightarrow{l} t'$ then $f(t) \xrightarrow{l} f(t')$.' Similarly, we often write just T for $(T, \longrightarrow)$, or even (L, T) to indicate that $(T, \longrightarrow)$ is a transition system with label set L.

Example 2.2 Any subset $S \subseteq L^*$ of lists (or 'words') over L gives rise to a transition system $(S, \longrightarrow)$, where $w \xrightarrow{l} w'$ iff $w' = wl$. For example, take $S = \{\varepsilon, a, aa, ab\} \subseteq \{a, b\}^*$, where ε is the empty list. Then we have $\varepsilon \xrightarrow{a} a$, $a \xrightarrow{a} aa$, and $a \xrightarrow{b} ab$, describing a simple transition system with a 'fork' at state a.

Any morphism $f : (S, \longrightarrow) \to (T, \longrightarrow)$ describes a similarly forking path (or *run*) in T: the definition of morphism requires both

$$f(\varepsilon) \xrightarrow{a} f(a) \xrightarrow{a} f(aa) \text{ and}$$
$$f(\varepsilon) \xrightarrow{a} f(a) \xrightarrow{b} f(ab)$$

(see diagram, right).



Example 2.3 Let $L = \{\mathsf{next}\}$, and consider a transition system $\mathit{Channel} = (L, \omega)$, where ω is the set of natural numbers, and the transition relation is universal: $m \xrightarrow{\mathsf{next}} n$ for all $m, n \in \omega$. We can think of this as a channel giving a stream of natural numbers: state $m \in \omega$ represents a channel where the first value in the stream is the number m, and the transition $m \xrightarrow{\mathsf{next}} n$ represents a destructive read, moving on to a state where the next value in the stream is n.

Similarly, a channel with a (one-place) buffer that determines the next value to be read on the channel would have $L = \omega \cup \{\mathsf{next}\}$, and could be defined as $\mathit{Sender} = (L, \omega \times \omega)$, where $\longrightarrow$ is the least relation with

$$(m_1, m_2) \xrightarrow{n} (n, m_2) \text{ for } n \in \omega$$
$$(m_1, m_2) \xrightarrow{\mathsf{next}} (m_1, m_1)$$

Thus, a transition with label $n \in \omega$ represents reading the value n into the buffer; and a next-transition copies that value to the channel.

Often, we are not so much interested in individual transitions between states, as in paths through a transition system; i.e., the sequences of actions that are allowed by the system. In particular, any transition system over L extends to a transition system over L^* with

$$t \xrightarrow{\varepsilon} t' \text{ iff } t = t'$$

$$t \xrightarrow{wl} t' \text{ iff } t \xrightarrow{w} t'' \text{ and } t'' \xrightarrow{l} t' \text{ for some } t'' \ .$$

That is, transitions can be freely extended to paths of transitions. We can use this to provide a slightly more general notion of transition system that takes labels in a monoid.

Definition 2.4 Let $\mathcal{M} = (M, \cdot, \varepsilon)$ be a monoid (we generally write mm' in place of $m \cdot m'$); we say m is a **prefix** of m', and write $m \leq m'$, iff $m' = mn$ for some $n \in M$, and we say that a subset $X \subseteq M$ is **prefix-closed** iff $x \leq y$ and $y \in X$ implies $x \in X$. We write $\Omega(\mathcal{M})$ for the set of all prefix-closed subsets of M (including M itself), and for $m \in M$, we write $m{\downarrow}$ for the set of all prefixes of m (including m itself)

A **labelled transition system over** $\mathcal{M}$ is a pair $(T, \longrightarrow)$, with $\longrightarrow \subseteq T \times M \times T$ such that

$$t \xrightarrow{\varepsilon} t' \text{ iff } t = t'$$
$$t \xrightarrow{mn} t' \text{ iff } t \xrightarrow{m} t'' \text{ and } t'' \xrightarrow{n} t' \text{ for some } t'' \in T \ .$$

A morphism $f : (T, \longrightarrow) \to (T', \longrightarrow)$ of transition systems over $\mathcal{M}$ is a function $f : T \to T'$ such that $t \xrightarrow{m} t'$ implies $f(t) \xrightarrow{m} f(t')$. This gives a category $\mathsf{LTS}_{\mathcal{M}}$ of transition systems over $\mathcal{M}$.

Very often, we want to relate transition systems with different labels sets.

Definition 2.5 Let T be a transition system over $\mathcal{M}$ and let T' be a transition system over $\mathcal{M}'$. A **morphism** $(\mathcal{M}, T) \to (\mathcal{M}', T')$ consists of a monoid homomorphism $h : \mathcal{M} \to \mathcal{M}'$ (i.e, $h(\varepsilon) = \varepsilon$ and $h(mn) = (h(m))(h(n))$) and a function $f : T \to T'$ such that $f(t_1) \xrightarrow{h(m)} f(t_2)$ whenever $t_1 \xrightarrow{m} t_2$.

Example 2.6 Example 2.3 defined a *Channel* and a *Sender* that added a one-place buffer to the channel. We can express this 'adding of a buffer' by a morphism from *Sender* to *Channel* that forgets about the buffer. Specifically, we can define $h_2 : (\omega \cup \{\mathsf{next}\})^* \to \{\mathsf{next}\}^*$ by, for $n \in \omega$:

$$n \mapsto \varepsilon$$
$$\mathsf{next} \mapsto \mathsf{next}$$

That is, h_2 forgets about all the numbers in a list, and keeps only the nexts; if we think about this as a mapping of paths in a transition system, the mapping ignores everything but next-transitions. Then $(h_2, \pi_2) : \mathit{Sender} \to \mathit{Channel}$.

As another example, we consider a process that reads values from a channel and stores some of the values read in a list. The state set is $\omega \times \omega^*$, where the first component of a pair represents a value read on a channel, and the second component represents the stored list of read values. Thus, we let $\mathit{Adder} = (\{\mathsf{next}, \mathsf{add}\}^*, \omega \times \omega^*)$ be the transition system with transitions defined by

$$(m, w) \xrightarrow{\mathsf{next}} (n, w)$$
$$(m, w) \xrightarrow{\mathsf{add}} (m, m\, w)$$

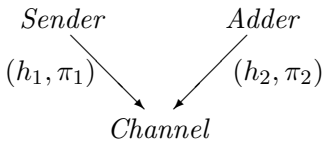


Fig. 1. Two objects with a shared subcomponent

(where $m\,w$ is the list w with m added to the front). In words, a next-transition reads a value on the channel, and an add-transition takes the current value on the channel and stores it in the list. Again, we can let $h_1 : \{\mathsf{next}, \mathsf{add}\}^* \to \{\mathsf{next}\}^*$ be defined by:

$$\mathsf{next} \mapsto \mathsf{next}$$
$$\mathsf{add} \mapsto \varepsilon$$

Then $(h_1, \pi_1) : \mathit{Adder} \to \mathit{Channel}$.

This example shows two systems that extend the *Channel* system in different ways. We might picture the situation as in Figure 1. We can give a system that combines the *Adder* and *Sender* systems in such a way that they share a common channel by taking the *limit* of this diagram. Rather than give a formal definition of the limit construction, we simply illustrate the concept by giving the construction for this particular example (see [22] for technical details). The monoid of labels is calculated as follows. Let $M = \{(x, y) \in (\omega \cup \{\mathsf{next}\})^* \times \{\mathsf{next}, \mathsf{add}\}^* \mid h_2(x) = h_1(y)\}$ which is a monoid with $\varepsilon = (\varepsilon, \varepsilon)$ and $(x, y)(x', y') = (x\,x',\ y\,y')$. Thus, the labels consist of *Adder*-labels and *Sender*-labels that agree on the labels for the shared channel. Note that 'local' actions (i.e., actions with no nexts) can occur in any order; for example:

$$(3, \varepsilon)(\varepsilon, \mathsf{add}) = (3, \mathsf{add}) = (\varepsilon, \mathsf{add})(3, \varepsilon) \ .$$

This gives the labels the structure of a Mazurkiewicz trace [23], where 'independent' actions are commutative. The equation above says that putting the value 3 in the buffer and adding the current value of the stream to the list of values read are independent actions, and the order in which they are performed is immaterial.

The states of the limit transition system are pairs consisting of a *Sender*-state (m_1, m_2) and an *Adder*-state (n, w) such that $\pi_2(m_1, m_2) = \pi_1(n, w)$, i.e., such that $m_2 = n$. Thus, the states are just $T = \omega \times \omega \times \omega^*$. Transitions are given by $(m, n, w) \xrightarrow{(x,y)} (m', n', w')$ iff $(m, n) \xrightarrow{x} (m', n')$ and $(n, w) \xrightarrow{y} (n', w')$. For example, we have $(0, 0, \varepsilon) \xrightarrow{z} (3, 2, [2])$, where $z = (2\,\mathsf{next}\,3,\ \mathsf{next}\,\mathsf{add})$, corresponding to placing 2 in the buffer, reading from the channel, adding the value read to the list, and placing 3 in the buffer. As described above, these last two, 'local' actions can be viewed as taking place in any order.

2.2 Sheaves

Sheaf theory is used in many branches of mathematics, the underlying theme in its various applications being the passage from local to global properties [17]. It

provides a formal notion of coherent system of observations: a number of consistent observations of various aspects of an object can be uniquely pasted together to give an observation covering all of those aspects. The passage from local to global properties, and the pasting together of local observations of behaviour allow sheaf theory to be usefully applied in computer science, to give models for concurrent processes [25,4,19] and objects [11,5,27,2,20]. We give a basic definition of 'sheaf' below; fuller accounts can be found in [26,18].

We may consider a sheaf as giving a set of observations of an object's behaviour from a variety of 'locations'. The notion of location is formalised by the following

Definition 2.7 A **complete Heyting algebra** is a partially ordered set $(C, \leq)$ such that:

- for all $c, d \in C$, there is a greatest lower bound $c \wedge d$
- for all subsets $\{c_i \mid i \in I\}$ of C, there is a least upper bound $\bigvee_{i \in I} c_i$
- greatest lower bounds distribute through least upper bounds:

$$(\bigvee_{i \in I} c_i) \wedge d \;=\; \bigvee_{i \in I} (c_i \wedge d) \;.$$

For example, the open sets of any topological space with the subset inclusion ordering give a complete Heyting algebra; also, any complete lattice is a complete Heyting algebra. In particular, the set of prefix-closed subsets of a monoid, $\Omega(\mathcal{M})$, is a complete Heyting algebra.

Definition 2.8 Let C be a complete Heyting algebra; a **presheaf** F on C is a functor from C^{op} to Set. That is, for each $c \in C$ there is a set $F(c)$, and for $c, d \in C$ such that $c \leq d$, there is a **restriction function** $F_{c \leq d} : F(d) \to F(c)$, subject to the following conditions:

- $F_{c \leq c} = id_{F(c)}$, the identity on the set $F(c)$; and
- if $c \leq d \leq e$, then $F_{c \leq d} \circ F_{d \leq e} = F_{c \leq e}$.

For a presheaf F on C, if $c \leq d$ in C and $x \in F(d)$, we often write $x\restriction_c$ for $F_{c \leq d}(x)$.

A sheaf is a presheaf which allows families of consistent local observations to be pasted together to give a global observation.

Definition 2.9 A presheaf F is a **sheaf** iff it satisfies the following **pasting condition**:

- if $c = \bigvee_{i \in I} c_i$ and $x_i \in F(c_i)$ is a family of elements for $i \in I$ such that $x_i\restriction_{c_i \wedge c_j} = x_j\restriction_{c_i \wedge c_j}$ for all $i, j \in I$, then there is a unique $x \in F(c)$ such that $x\restriction_{c_i} = x_i$ for all $i \in I$.

A morphism of sheaves $\theta : F \to G$ is just a natural transformation from F to G viewed as presheaves; i.e., θ is a family of functions $\theta_c : F(c) \to G(c)$ for $c \in C$ that respect restrictions: given $c \leq d$, and $x \in F(d)$,

$$(\theta_d(x))\restriction_d = \theta_c(x\restriction_c) \;.$$

Since we are particularly interested in sheaves over the complete Heyting algebra $\Omega(\mathcal{M})$ for a monoid $\mathcal{M}$, we write $\mathsf{Sh}_{\mathcal{M}}$ for the category of sheaves over $\Omega(\mathcal{M})$.

Cognoscenti will recognise in the pasting condition the statement that a sheaf has $F(c)$ as a limit of all $F(c_i)$ for coverings $c = \bigvee_{i\in I} c_i$. In other words, elements of $F(c)$ are constructed as limits of approximations given by consistent families $(f_i \in F(c_i))_{i\in I}$

Example 2.10 Let T be a topological space, and for every open set X, let $F(X)$ be the set of continuous real-valued functions $f : X \to R$. If $X = \bigcup_{i\in I} X_i$, and we have $f_i \in F(X_i)$ for each i, then this family of functions can be pasted uniquely together to form $f \in F(X)$ (i.e., a continuous $f : X \to R$), provided that the f_i agree on overlaps: i.e., provided that $f_i{\restriction}_{X_i\cap X_j} = f_j{\restriction}_{X_i\cap X_j}$ for all $i, j \in I$.

The following gives another illustrative example. One motivation for this example is a constructive notion of truth that, instead of asking 'is this true?', asks 'how true is this?' For example, any given function defined on the points of a topological space may or may not be continuous; but, rather than ask *whether* the function is continuous, we might ask *how* continuous it is, and an appropriate reply might consist of detailing all the open sets where the function actually is continuous — and if the answer is all open sets, then the function is everywhere continuous. Thus, subsets of the open sets of a topology can serve as generalised truth-values.

Example 2.11 For $S \subseteq M$, if we write $\Omega(S)$ for the prefix-closed subsets of S, then Ω is a sheaf over $\Omega(\mathcal{M})$; given an inclusion $X \subseteq Y$ of prefix closed sets, then $\Omega_{X\subseteq Y}$ takes $V \subseteq Y$ (i.e., $V \in \Omega(Y)$) to $V\cap X \subseteq X$ (i.e., $V\cap X \in \Omega(X)$).

Again, cognoscenti will recognise the subobject classifier of the topos of sheaves; for the purposes of the present paper, let us point out that this notion of the degree of satisfaction of some property (such as continuity) is a good example of what we have loosely termed an 'observation', and the fact that this is a sheaf gives some credence to our slogan that sheaves are coherent systems of observations.

As a further example of sheaves, the remainder of this subsection will show that labelled transition systems are sheaves. The following is a somewhat simplified version of results presented in [22].

Given a transition system T, we construct a sheaf from T by considering sets of paths in T, as in Example 2.2. Recall that a (forking) path in T was just a transition system morphism $f : X \to T$ for some $X \in \Omega(\mathcal{M})$. If we have $f_1 : X_1 \to T$ and $f_2 : X_2 \to T$ such that $f_1{\restriction}_{X_1\cap X_2} = f_1{\restriction}_{X_1\cap X_2}$, then clearly these functions can be uniquely pasted to give a morphism, or path, $X_1\cup X_2 \to T$. Thus, for every transition system T, we have a sheaf $\mathbf{Sh}_{\mathcal{M}}(T)$, which is defined by

$$\mathbf{Sh}_{\mathcal{M}}(T)(X) = \mathsf{LTS}_{\mathcal{M}}(X, T) \ ,$$

where $\mathsf{LTS}_{\mathcal{M}}(X, T)$ is the set of transition system morphisms from X to T for any prefix-closed subset $X \in \Omega(\mathcal{M})$ (cf. Example 2.2). Note that, because every $X \in \Omega(\mathcal{M})$ is a transition system, $\mathsf{LTS}_{\mathcal{M}}(\_, T)$ is a functor $\Omega(\mathcal{M})^{\mathrm{op}} \to \mathsf{Set}$, and so this definition also applies for morphisms (i.e., inclusions) in $\Omega(\mathcal{M})$. That is,

restriction in $\mathbf{Sh}_{\mathcal{M}}(T)$ is restriction of paths.

Moreover, any morphism of transition systems $f : T \to U$ gives a morphism of sheaves

$$\mathbf{Sh}_{\mathcal{M}}(f) : \mathbf{Sh}_{\mathcal{M}}(T) \to \mathbf{Sh}_{\mathcal{M}}(U)$$

defined by saying that for each $X \in \Omega(\mathcal{M})$, the component $\mathbf{Sh}_{\mathcal{M}}(f)_X$ takes a T-path $h : X \to T$ to the U-path $f \circ h : X \to U$.

Going the other way, we can construct a transition system from any sheaf. We can represent a sheaf F by its set of 'elements' (m, e), where $m \in M$ and $e \in F(m{\downarrow})$. Transitions on these states are given by the restriction actions of F. That is, the transition system $\mathbf{Tr}_{\mathcal{M}}(F)$ generated by sheaf F is defined by

$$\mathbf{Tr}_{\mathcal{M}}(F) = \sum_{m \in M} F(m{\downarrow}) \ ,$$

and transitions in this system are defined by $(m, e) \xrightarrow{n} (m', e')$ iff $m' = mn$ and $e'{\restriction}_{m\downarrow} = e$.

Moreover, a morphism of sheaves $\theta : E \to F$ gives a morphism of transition systems

$$\mathbf{Tr}_{\mathcal{M}}(\theta) : \mathbf{Tr}_{\mathcal{M}}(E) \to \mathbf{Tr}_{\mathcal{M}}(F)$$

taking $(m, e) \in \mathbf{Tr}_{\mathcal{M}}(E)$ to $(m, \theta_{m\downarrow}(e)) \in \mathbf{Tr}_{\mathcal{M}}(F)$.

The upshot of all of this is not to say that transition systems *are* sheaves: technically, as reported in [22], tree-unfoldings of transition systems with distinguished initial states are sheaves. Yet more technically: there is an adjunction between sheaves and transition systems, and this is a reflection when distinguished initial states are taken into consideration. These technical results mean that limits of transition systems are the same as limits of the corresponding sheaves, so that sheaves can be thought of as the compositional semantics of transition systems — 'compositional' in the sense that building transition systems hierarchically by means of limit constructions, as illustrated here, gives the same results whether we take limits of transition systems or of their underlying sheaves. In the following subsection, we extend this hierarchical approach to algebraic specifications of systems.

2.3 Component Specifications

We now consider a third way of specifying objects: hidden algebra. Hidden algebra was developed by Goguen [10] as a semantic foundation for the object paradigm. It is a variant of many-sorted algebraic specification with the key feature that sorts are divided into 'hidden' and 'visible' sorts. Visible sorts are intended to represent data values such as numbers, boolean values, and so on, while hidden sorts represent states of objects. In other words, visible sorts represent immutable data values, while hidden sorts represent mutable states that change under the action of the operations in the specification. A comprehensive introduction to hidden algebra is given in [14,15].

Goguen and Diaconescu [12] describe a way of combining hidden algebraic specifications in a way that captures parallel composition of objects with shared subcomponents, which they call concurrent connection. It turns out that it is possible

to see concurrent connection as yet another example of a limit construction. As usual, we present this construction by means of examples; the technical details are given in [21].

Let's suppose that we have a specification, `DATA`, of some data structures: the examples below assume these include at least a sort `Nat` of natural numbers, and `NatList` of lists of natural numbers. The hidden specifications that we look at below will extend this specification with one hidden sort, representing the states of some object, and a number of different operations that change those states. The simplest example simply declares one hidden sort; we use the notation of the hidden algebraic specification language BOBJ [13]:

```
bth STATE is pr DATA .
   sort State .
end
```

Here, `STATE` is the name of the specification, `pr DATA` imports the specification of data values, and the line `sort State` declares a hidden sort called `State`. This hidden sort represents the states of some object — though not a particularly interesting object, as there are no operations for changing these states!

A more interesting example, specifying a channel as in Example 2.3, is given below. The specification `STATE` is extended with two operations: the first, `next`, changes the state, notionally by reading a value from the stream; the second, `val`, returns a natural number, notionally the number that is read from the stream. We adopt an object-oriented terminology in calling operations (such as `next`) that return hidden sorts 'methods', and operations (such as `val`) that return visible sorts 'attributes'.

```
bth CHANNEL is pr STATE .
   op next : State -> State .
   op val : State -> Nat .
end
```

Thus, given a state `s`, the successive values that can be read on the channel are given by

```
val(s), val(next(s)), val(next(next(s))), val(next(next(next(s))))
```

and so forth. Note that the absence of equations in this specification means that these values are not constrained in any way (apart from the type constraint that they are all natural numbers); in this way, hidden algebra provides an elegant way of capturing non-determinism (see [14]).

We can extend this specification again to obtain a one-place buffer with the same behaviour as the *Sender* transition system of Example 2.3. In that example, we used natural numbers n as labels for transitions that placed the value n in the buffer; here we introduce an operation `put`, that takes the value n as argument. The operation `store` returns the value that is currently stored in the buffer; that

is, in a state `s`, `store(s)` represents the value in the buffer.

```
bth SENDER is pr CHANNEL .

   op put : State Nat -> State .
   op store : State -> Nat .

   var S : State .
   var M : Nat .

   eq  store(put(S,M))  =  M .
   eq  store(next(S))  =  store(S) .

   eq  val(put(S,M))  =  val(S) .
   eq  val(next(S))  =  store(S) .

end
```

The first equation says that `put` sets the value of the buffer; the second says that reading from the channel has no effect on the contents of the buffer; the third says that setting the value in the buffer does not affect the current value in the channel; whereas the fourth says that after a destructive read, the value in the buffer is put on the stream.

In a similar way, the *Adder* object of Example 2.6 can be specified as follows (we use `cons` for the operation of adding an element to a list):

```
bth ADDER is pr CHANNEL .

   op total : State -> NatList .

   var S : State .
   eq  total(next(S))  =  cons(val(S), total(S)) .
end
```

Now we have two specifications, `SENDER` and `ADDER`, of objects that have a channel as a subcomponent. The concurrent connection of these two specifications represents an object composed of a sender and an adder that communicate through their shared channel: values written to the buffer are sent across the channel to the adder. In general, the concurrent connection has all the operations and equations from the component specifications, without duplicating the operations and equations from shared subcomponents. There are also 'independence axioms' that state that methods that are local to one specification do not affect attributes that are local to the other specification. The concurrent connection of `SENDER` and `ADDER` is

```
bth SENDER-ADDER is pr CHANNEL .

   op put : State Nat -> State .
   op store : State -> Nat .

   op total : State -> NatList .

   var S : State .
   var M : Nat .

   eq  store(put(S,M))  =  M .
```

```
    eq  store(next(S))  =  store(S) .

    eq  val(put(S,M))  =  val(S) .
    eq  val(next(S))  =  store(S) .

    eq  total(next(S))  =  cons(val(S), total(S)) .

    eq  total(put(S,M))  =  total(S) .

end
```

Here, the final equation is the independence axiom. There is only one local method in the `SENDER` specification, namely `put`, and only one local attribute in the `ADDER` specification; the equation

```
    eq  total(put(S,M))  =  total(S) .
```

states that the method `put`, which is local to the sender, has no effect on the attribute `total`, which is local to the adder.

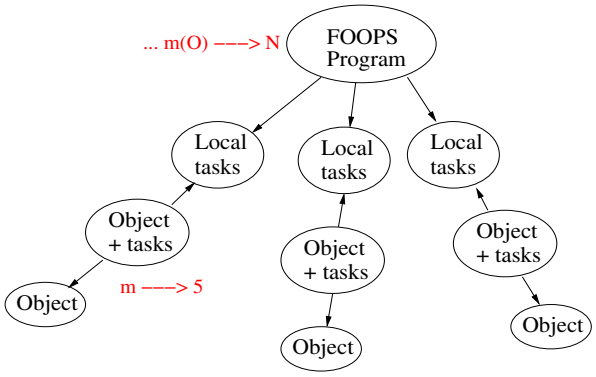
This example suggests that concurrent connection captures parallel composition with communication through shared subcomponents in the same way as the limits of transition systems described in Section 2.1. Indeed, our previous work in [21] shows that, provided that subcomponents are extended in a particular way (in brief: local methods do not affect the values of the subcomponent's attributes), then concurrent connection is an example of a limit construction.

We have seen that labelled transition systems can be viewed as sheaves. It would be possible to give an analogous treatment of the hidden algebraic specifications described above. However, at this point we prefer to concentrate on the hierarchical approach of building complex systems from component systems that is common to hidden algebra and labelled transition systems, and works in both cases by limit constructions. It turns out that this also gives us yet another example of sheaves.

3 Dynamic Systems

In [2], Cîrstea gave a sheaf-theoretical semantics for an object-oriented language. This semantics specified objects as sheaves that co-operated concurrently to evaluate a declarative program. The behaviour of this ensemble of objects was given as a limit construction for a diagram of the following form, where each oval represents a sheaf. One limitation of this approach is that it is assumed that all possible objects are already present in the diagram: there was no notion of object creation or destruction, and no dynamic structuring of objects, whereby one object could become a subobject of an amalgamated object (such as a node added to a linked

list).



In this final section, we look at how sheaves can represent hierarchical structure, and how they might shape future research into the semantics of object-oriented languages that take account of the dynamic aspects of systems of interacting objects.

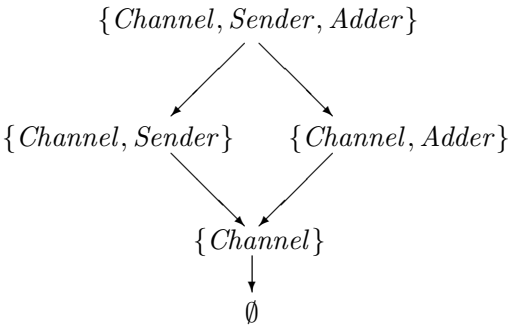
We begin with a generalisation of the notion of set-valued sheaf. The definition of sheaf in Section 2.2 was followed by an observation that sheaves captured the idea of limits of approximations. The following generalisation of set-valued sheaves is due to Gray [16]. Its statement requires considerably more knowledge of category theory than we've assumed in this paper; a more intuitive statement is that a sheaf provides observations that are constructed as limits of observations of subcomponents.

Definition 3.1 A **sheaf with values in a category** L is a functor F from a complete Heyting algebra to L such that if $X = \bigvee_{i\in I} X_i$, then

$$F(X) \longrightarrow \prod_{i\in I} F(X_i) \quad \Longrightarrow \prod_{i,j\in I} F(X_i \wedge X_j)$$

is an equaliser diagram (where all the arrows arise from the obvious restrictions by the universal property of the target product).

How does this relate to the hierarchical systems of the previous section? Consider again the situation pictured in Figure 1, where the *Channel* is a subcomponent of both the *Sender* and *Adder* objects. We can think of this subcomponent relation as an ordering on the objects, so that $Channel < Sender$ and $Channel < Adder$. Now the downwards-closed subsets of objects is a lattice (and so a complete Heyting algebra):



An example of a functor, F, from this complete Heyting algebra to the category of labelled transition systems assigns:

- to $\{Channel\}$ the *Channel* object of Example 2.3;
- to $\{Channel, Sender\}$ the *Sender* object from the same example;
- to $\{Channel, Adder\}$ the *Adder* object of Example 2.6; and
- to $\{Channel, Sender, Adder\}$ the limit object, *Sender-Adder*, constructed at the end of Section 2.1.

Now to say that F is in fact a sheaf of labelled transition systems is precisely to say that *Sender-Adder* is constructed as a limit from *Sender* and *Adder*.

In general, if limits exist in the category L, then functors that assign objects to diagrams such as that in Figure 1 can be extended to sheaves on diagrams such as that above. Let LTS be the category of labelled transition systems, and let $\Omega(\mathsf{X})$ be the category of downwards-closed subsets of a preorder X:

Proposition 3.2 *Let X be a preorder category, and let $\delta : \mathsf{X} \to \mathsf{LTS}$. Define $\delta^* : \Omega(\mathsf{X}) \to \mathsf{LTS}$ by $\delta^*(X) = \lim(\delta\restriction_X)$; then δ^* is a sheaf of transition systems.*

Note that the prefix-closed sets we used in Section 2.1 are examples of the downwards-closed sets, which is why we use the notation $\Omega(\mathsf{X})$ for downwards-closed sets in this proposition. Slightly more generally, we can extend this to any diagram presented as a graph:

Proposition 3.3 *Let G be a directed graph, and let $\mathcal{O}(G)$ be the subsets of nodes closed 'under outgoing edges'.*

Then $\delta : G \to L$ gives $\delta^* : \mathcal{O}(G) \to L$, where

$$\delta^*(X) = \lim(\delta\restriction_X) \ .$$

Limits exist for labelled transition systems, and also for the hidden algebraic specification considered in Section 2.3, so if any graph is decorated with labelled transition systems, or with hidden algebraic specifications, then that decoration extends to a sheaf of labelled transition systems or specifications, and even models of those specifications. (In fact, there are restrictions on the form of the specifications and morphisms between them; the interested reader is directed to [21] for details.) In conjunction with the slogan 'behaviour is limit', the definition of a sheaf of transition systems means that such sheaves describe the behaviour of hierarchically structured systems.

What about the dynamic aspects of systems of interacting objects? We end with a sketch of some constructions that allow modelling systems with changing topologies. Research by Colman Reilly at University College Dublin suggests looking at systems whose state comprises a graph that represents the topology of a system. We start by looking at the simplest examples of these, then add progressively more structure, so that we can look at sheaves of sets, then sheaves of transition systems, then sheaves of hidden algebraic specifications.

For sheaves of sets, we specify a state set consisting of tuples (G, δ, b), where

- G is a graph,
- $\delta : G \to \mathsf{Set}$, and
- b is an element of the *limit* of δ (i.e., an element of δ^*).

That is, states are graphs decorated with sets, and also with elements of these sets in a coherent way; that is the import of looking at the sheaf of sets δ^*. The idea is to think of these states as snapshots of an evolving system: the graph describes the current topology of the system; for each node n, the set $\delta(n)$ is the state set of the object at that node; and $\delta^*(n{\downarrow})$ picks out the actual state of that object, which will be the limit of all the subcomponents in the subgraph $n{\downarrow}$.

How might such a system evolve? A minimum requirement would be that some part of the system remains constant, while objects outside that constant part disappear, and new objects may be created. Thus, we allow transitions $(G, \delta, b) \longmapsto (G', \delta', b')$ iff

- there is a span $G \hookleftarrow G_0 \hookrightarrow G'$, and
- $\delta{\restriction}_{G_0} = \delta'{\restriction}_{G_0}$.
- $b{\restriction}_{G_0} = b'{\restriction}_{G_0}$.

These three conditions state that G_0 is the unchanged part of the system; outside of G_0, the system is allowed to evolve by adding or deleting nodes, provided that there is still some coherent state b' that agrees with the states of G_0.

This is somewhat static, as the changes that occur are objects outside G_0 disappearing from G, and new objects appearing in G', but doesn't allow the objects in G_0 to change their state. Instead of having a sheaf of sets, we may posit the snapshots of the system being given by sheaves of labelled transition systems. That is, the state set consists of tuples (G, δ, b), where

- G is a graph,
- $\delta : G \to \mathsf{LTS}$, and
- b is an element of the *limit* of δ (i.e., an element of δ^*).

Now the object decorating a node n is a labelled transition system, with states $b_n \in \delta^*(n{\downarrow})$ constructed as limits of the states of the transitions systems labelling the subsystems in $n{\downarrow}$.

As before, we posit that transitions of this system identify a constant part, with objects appearing and disappearing outside of it. We also allow the fixed part to evolve according to their local transitions: each transition system $\delta(n)$ can evolve independently. That is, the transitions of the system are: $(G, \delta, b) \stackrel{l}{\longmapsto} (G', \delta', b')$ iff

- there is a span $G \hookleftarrow G_0 \hookrightarrow G'$,
- $\delta{\restriction}_{G_0} = \delta'{\restriction}_{G_0}$, and
- $b{\restriction}_{G_0} \stackrel{l{\restriction}_{G_0}}{\longmapsto} b'{\restriction}_{G_0}$ in $Lim(\delta{\restriction}_{G_0})$.
- $l \in labels(Lim(\delta{\restriction}_{G_0}))$.

In other words, G_0 is the constant part of the system, whose behaviour is described by the limit of δ restricted to G_0. The label, l, of the transition comes from the labels of that limiting object. For each node n in G_0, this gives a label l_n for the transition system at n, and we allow these states to evolve by $b_n \xrightarrow{l_n} b'_n$.

Going one stage further, we might consider systems whose components are specified by hidden algebraic specifications, and whose topologies may change by objects combining via concurrent connection. For example, a `SENDER` object and an `ADDER` object might combine into a `SENDER-ADDER` object by creating a shared `CHANNEL`. Thus, the state set would consist of tuples (G, δ, b), where

- G is a graph,
- δ assigns to nodes:
  - a hidden theory T, and
  - a model B of T, and
- b is an element of the limit of δ.

The only change here is that we have sheaves of hidden specifications together with models of those specifications. The models provide the carrier sets whose elements are the local states of the objects $\delta(n)$.

The transitions of these systems are slightly more complex, since the local states would evolve by means of the operations in the corresponding local hidden theory, and these are interpreted by models as functions, which are, by definition, deterministic. Such a deterministic transition is given by the schema $(G, \delta, b) \xrightarrow{\sigma} (G, \delta, \delta^*(\sigma)(b))$ for σ in $lim(\delta)$. That is, the transition occurs by means of an operation of the limit theory, which is the concurrent connection of the entire graph, as in Section 2.3. As for transitions that change the topology of the system, by allowing objects to disappear, or new objects to be added, or by allowing new concurrent connections to be created, we posit $(G, \delta, b) \xrightarrow{\sigma} (G', \delta', b')$ with an invariant G_0 contained in G and G' as before.

Some details still need to be worked out, but the use of sheaves of various structures allows us to give standard constructions that apply to a variety of structures: the labelled transition systems and hidden theories considered here are just examples of the kinds of semantic models that can be used in modelling dynamic systems of interacting objects. Indeed, the only requirement on the kinds of models that can be used is the existence of limits.

We have concentrated here on the semantic foundations; i.e., we have been addressing the question of what models of dynamically changing, hierarchical systems of interacting objects look like, but there are several open questions relating to the syntactic side of such systems. Perhaps the most important of these is: what linguistic constructs would allow these systems to be specified? One imagines that graph-rewriting techniques could be used to specify transitions of such systems, and that these could be usefully combined with the notion of 'theories as types', as introduced in [10]. Developing such a language that combines hidden algebraic theories with graph rewriting would thus be an interesting project; another inter-

esting project would be to use the dynamic, hierarchic systems presented here to give semantics to existing languages. An obvious first step would be to modify the 'static' semantics of FOOPS given in [2] so as to allow creation, deletion and restructuring of relations between objects. One could then tackle various process calculi and programming languages that allow dynamically changing configurations of objects, or even less explicitly linguistic systems, such as protein interactions, as in [6].

There are other approaches to combining algebraic specifications to model hierarchical systems. The approach of Diaconescu [3] can be seen as a sort of dual to that of the present paper; Diaconescu proposes constructs for dynamic synchronisation of objects within what could be seen as a flattened version of the concurrent connection treated here. An open question would be to make precise the relationship between these 'external' and 'internal' approaches.

References

[1] Cattani, G. L. and G. Winskel, *Presheaf models for concurrency*, in: *Computer Science Logic: Tenth international Workshop, CSL'96, Annual Conference of the EACSL. Selected Papers*, number 1258 in Lecture Notes in Computer Science (1997), pp. 58–75.

[2] Cîrstea, C., *A distributed semantics for FOOPS*, Technical Report PRG-TR-20-95, Programming Research Group, University of Oxford (1995).

[3] Diaconescu, R., *Behavioural specification for hierarchical object composition*, Theoretical Computer Science **343** (2005), pp. 305–331.

[4] Dubey, R., "On a general definition of safety and liveness," Master's thesis, School of Electrical Engineering and Comp. Sci., Washington State Univ. (1991).

[5] Ehrich, H.-D., J. A. Goguen and A. Sernadas, *A categorial theory of objects as observed processes*, in: J. d. Bakker, W. d. Roever and G. Rozenberg, editors, *Foundations of Object Oriented Languages* (1991), pp. 203–228.

[6] Fisher, M. J., G. Malcolm and R. C. Paton, *Spatio-logical processes in intracellular signalling*, Biosystems **55** (2000), pp. 83–92.

[7] Goguen, J. A., *Systems and minimal realization*, in: *Proceedings, 1971 IEEE Conf. on Decision and Control*, 1972, pp. 42–46.

[8] Goguen, J. A., *Objects*, International Journal of General Systems **1** (1975), pp. 237–243.

[9] Goguen, J. A., *Complexity of hierarchically organized systems and the structure of musical experiences*, International Journal of General Systems **3** (1977), pp. 233–251.

[10] Goguen, J. A., *Types as theories*, in: G. M. Reed, A. W. Roscoe and R. F. Wachter, editors, *Topology and Category Theory in Computer Science*, Oxford University Press, 1991 pp. 357–390.

[11] Goguen, J. A., *Sheaf semantics for concurrent interacting objects*, Mathematical Structures in Computer Science **11** (1992), pp. 159–191.

[12] Goguen, J. A. and R. Diaconescu, *Towards an algebraic semantics for the object paradigm*, in: H. Ehrig and F. Orejas, editors, *Recent Trends in Data Type Specification* (1994), pp. 1–29.

[13] Goguen, J. A., K. Lin and G. Roşu, *Circular coinductive rewriting*, in: *Proceedings, Automated Software Engineering '00* (2000), pp. 123–131.

[14] Goguen, J. A. and G. Malcolm, *A hidden agenda*, Theoretical Computer Science **245** (2000), pp. 55–101.

[15] Goguen, J. A., G. Malcolm and T. Kemp, *A hidden Herbrand theorem: combining the functional, object and logic paradigms*, Journal of Logic and Algebraic Programming **51** (2002), pp. 1–41.

[16] Gray, J., *Sheaves with values in a category*, Topology **3** (1965), pp. 1–18.

[17] Gray, J., *Fragments of the history of sheaf theory*, in: M. Fourman, C. Mulvey and D. Scott, editors, *Applications of Sheaves* (1980), pp. 1–79.

[18] Lane, S. M. and I. Moerdijk, "Sheaves in Geometry and Logic," Springer-Verlag, 1992.

[19] Lilius, J., *A sheaf semantics for Petri nets*, Technical Report A23, Dept. of Computer Science, Helsinki University of Technology (1993).

[20] Malcolm, G., *Interconnection of object specifications*, in: S. Goldsack and S. Kent, editors, *Formal Methods and Object Technology*, Springer Workshops in Computing, 1996 pp. 205–226.

[21] Malcolm, G., *Component-based specification of distributed systems*, Electronic Notes in Theoretical Computer Science **160** (2006), pp. 211–224.

[22] Malcolm, G., *Sheaves and structures of transition systems*, in: *Algebra, Meaning and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of his 65th Birthday* (2006), pp. 405–419.

[23] Mazurkiewicz, A., *Traces, histories, graphs: instances of a process monoid*, in: M. Chytil and V. Koubek, editors, *Mathematical Foundations of Computer Science* (1984), pp. 115–133.

[24] Monteiro, L., *Observation systems*, Electronic Notes in Theoretical Computer Science **33** (2000).

[25] Monteiro, L. and F. Pereira, *A sheaf-theoretic model of concurrency*, in: *Proc. Logic in Computer Science (LICS '86)* (1986), pp. 66–76.

[26] Tennison, B., "Sheaf Theory," London Mathematical Society Lecture Notes **20**, Cambridge University Press, 1975.

[27] Wolfram, D. A. and J. A. Goguen, *A sheaf semantics for FOOPS expressions (extended abstract)*, in: M. Tokoro, O. Nierstrasz, P. Wegner and A. Yonezawa, editors, *Proceedings of the ECOOP'91 Workshop on Object-Based Concurrent Computing* (1992), pp. 81–98.