THE UNIVERSITY OF CALGARY

Building process categories

BY

David A. Spooner

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
July, 1997

Contents

Chapter 1

Introduction

In functional programming, the (denotations of) types and functions form the objects and arrows of a category. In this setting, the mechanism which enables computation — function application — is governed by composition in the underlying category. The benefits are...

The situation for concurrent programming is less satisfactory, with the dominating formalisms for concurrent processes (derivatives of CCS [27] and CSP [19]) remaining largely untyped. In a concurrent system computation corresponds to process *interaction* and it has been generally accepted that process interaction generalizes function composition [**?**]. Still the prevailing view in the concurrency community it that processes are objects in model categories, and this makes inaccessible the "propositions as types" analogy for concurrent programming.

The paradigm of Interaction Categories, proposed by Abramsky [1], aims to put concurrent programming on a semantic foundation similar to that for functional programming. An interaction category takes objects to be interface specifications and morphisms to be concurrent processes, with composition given by process interaction.

1.1 Purpose of the thesis

This thesis is a technical exploration in the construction of *process categories*: i.e. categories whose morphisms are concurrent processes (modulo some behavioral equivalence), with composition given by process interaction.

We aim to develop a general construction of process categories which gives an explicit treatment of process equivalence and which accounts for many of the features (such as asynchrony) found in interaction categories. The proposed techniques arise from the view that a process category should be built upon a category of models of concurrency, and (since process categories are inherently more complex) that reasoning about the structure of a process category should be done at the more basic level of the model category.

Another motivating factor in this development is one has significant latitude in the choice of a model category: our techniques provide a link between the state-based models, which admit finite expression of processes, and the more abstract models in which the mathematical properties of a process category are most easily established.

1.2 Organization of the thesis

Chapter 2 presents the construction of a process category as a span construction quotiented by a cover system: the underlying model category expresses the intended dynamics of processes, and the cover system provides the notion of behavioral equivalence on models. The main technical development is twofold: we examine various means to obtain cover systems on model categories, and we show that the process construction is 2-functorial. The latter fact is used to infer the presence of linear type structure in a process category from related structure in it's model category. We illustrate the techniques by constructing categories of synchronous processes upon models of transition systems and games.

In chapter 3 we present an approach to obtaining asynchrony in a process category: one takes a monad of delay on a synchronous model category, and then performs the process construction upon the Kleisli category of the delay monad. As Kleisli categories do not generally provide suitable model categories, the main technical development is the characterization of those those monads which support the process construction. We show how the monad of delay on transition systems can be used to reconstruct Abramsky's category ASProc of asynchronous processes. We also consider how to obtain suitable delay monads for categories of games.

In chapter 4 we show how adjunctions between model categories can induce equivalence of the associated process categories. We also provide a general characterization of behaviors in a model category: these correspond to coreflective subcategories which yeild the same category of processes as the entire model category. For illustration, we show that a wide range of models of interleaved concurrency give rise to the same process category — Abramsy's category SProc of synchronous processes.

Before concluding, chapter 5 shows that categories of circuits can provide a basis for programming in process categories. Beginning with a simply typed category of partial circuits, one can extract algebraically categories of circuits which satisfy certain safety and liveness properties. These circuits categories are shown to embed into SProc.

1.3 Related work

Abramsky's work on Interaction Categories has brought forth the idea that processes should form the morphisms of a category in which composition is given by process interaction. Many of the techniques presented in this thesis were motivated by a desire to account for features found in the key examples of interaction categories. One such example is the category SProc of synchronous processes (see [1] or [2]): it's objects are trace specifications, and its morphisms are (strong bisimulation equivalence classes of) transition systems whose traces lie within its specifications. Composition is given by a combination of "synchronous product" and "restriction", using the terminology of Milner's process calculus SCCS [26]. Distilling the presentation of [1], SProc is the category in which

- objects A are non-empty and prefix-closed sets $S_A \subseteq \Sigma_A^*$ of strings over alphabet Σ_A;
- morphisms $p : A \to B$ are given by the synchronization trees over alphabet $\Sigma_A \times \Sigma_B$ which satisfy $\mathrm{traces}(p) \subseteq S_A \times S_B$, where
  - · the synchronization trees over a label set Σ are the largest solution to the (non-well-founded) set equation $\mathrm{ST}_\Sigma = \mathcal{P}(\Sigma \times \mathrm{ST}_\Sigma)$, and
  - · $\mathrm{traces}(p) = \{\, [\,] \,\} \cup \{\, a :: \ell \mid \exists q.\ (a, q) \in p \ \wedge\ \ell \in \mathrm{traces}(q)\}$
- composition of $p : A \to B$ and $q : B \to C$ is given by

$$p; q \stackrel{def}{=} \{\ ((a, c),\ p'; q') \mid \exists b.\ ((a, b), p') \in p \ \wedge\ ((b, c), q') \in q\ \}$$

- identites are given by

$$1_A = \{\ ((a, a), 1_{A/a}) \mid a \in S_A\ \}$$

  where $A/a \stackrel{def}{=} \{\ \ell \mid a :: \ell \in S_A\ \}$.

Note that it is common practice in presenting interaction categories to choose canonical representations for processes, thus obviating the issue of process equivalence.

Joyal, Nielsen and Winskel [21] have shown that *open morphisms* (which we call cover systems) can be used to express standard notions of equivalence in a wide range of model categories for concurrency. The general form is that objects A and B in a model category are $\mathcal{X}$-bisimilar when related by a span

$$\begin{tikzcd} & C \arrow[dl, "f"'] \arrow[dr, "g"] & \\ A & & B \end{tikzcd}$$

of $\mathcal{X}$-open morphisms. Concerning bisimulation in the category of nondeterministic transition systems with fixed label set Σ, the open morphisms are those which reflect transitions: $f : C \to A$ is open if it has the property that for any state s in C and every transition $f(s) \xrightarrow{a} t$ in A there exists a transition $s \xrightarrow{a} s'$ in C such that $f(s') = t$. Two objects in this category are strong bisimilar exactly when related by a span of such open morphisms [21]. Our interest in cover systems is that they provide exactly the structure necessary to quotient a bicategory of spans down to a category.

Nielsen, Sassone and Winskel have established relationships in the form of adjunctions amongst many of the standard model categories for concurrency. In general, one has the following criteria for classification:

- *How the state-space is modeled.* The *system* models, such as transition systems, allow a process to reach the same state repeatedly through its evolution, while the *behavior* models, such as trees, introduce maximal separation in the state space so that each state determines the process history at that point.
- *Whether nondeterminism is modeled.* The *branching time* models, such as synchronization trees, model internal/nondeterministic choice, while the *linear time* models, such as traces, are deterministic.

It is shown that the behavior models form coreflective subcategories of the associated system models, and that the linear-time models are reflective subcategories of the associated branching-time models. For the standard models of interleaved concurrency one has the following picture:

$$\begin{tikzcd} \mathrm{Trace} \arrow[r, tail] \arrow[d, tail] & \mathrm{STree} \arrow[d, tail] \\ \mathrm{Tran} \arrow[r, tail] & \mathrm{NTran} \end{tikzcd}$$

where horizontal arrows are reflections and vertical arrows are coreflections. We show that, for certain cover systems, these relationships pass through the process construction and in fact become adjoint equivalences.

Another approach to constructing categories of processes are the bicategories of circuits of Katis, Sabadini and Walters (see [23] or [22]): the bicategory Circ(**Set**) has

- objects A are sets
- 1-cells $A \to B$ are given by pairs $(S_f, f : A{\times}S_f \to S_f{\times}B)$ of state spaces and transition functions, with composition $f;g$ given (ommiting associativity) by $(S_f{\times}S_g,\ f{\times}1; 1{\times}g)$
- 2-cells $f \Rightarrow g : A \to B$ are functions α satisfying $f; \alpha{\times}1 = 1{\times}\alpha; g$, with (vertical) composition given in **Set**.

Circuit equivalence is given essentially by trace equivalence, obtained through a morphisms of bicategories into Span(**Set**). We show that a related category of circuits can be embedded into SProc, although (given theorem 4.4.1 of chapter 4) the behavior functor of [23] is essentially a functor into SProc already.

1.4 Contributions of the thesis

We present the construction of a process category from a category of process models. This construction offers several benefits in the formulation of interaction categories:

- the explicit treatment of equality facilitates the use of state-based notation and thus finite presentation of processes;
- as equivalence is given by a cover system, one can easily vary the equivalence on the processes generated from a specific model category;
- one can calculate at the level of a model category to establish the structure of a (more complicated) process category.

We have shown that Abramsky's category SProc of synchronous processes modulo bisimulation arises through our construction as a process category upon all of the standard models of interleaved concurrency. Furthermore, SProc modulo trace equivalence is the category of relations on the category of trees: as trees are essentially "sets in time", this result sheds new light on the analogy of SProc and "relations in time". This work is reported in [15].

We have shown, contrary to suggestions in [2], that the Fock construction [9] cannot be used to model storage in SProc. In retrospect we believe that the original formulation of SProc solicits excessive intuition from the category of sets and relations, and that such a conclusion cannot be drawn when viewing storage as arising in the model category.

We have developed techniques for introducing asynchrony into a synchronous process category, which resembles Milner's progression from synchrony to asynchrony in process calculi. Using these techniques, we shown how Abramsky's category ASProc of asynchronous processes modulo weak bisimulation arises from the category SProc of synchronous processes. As a result of this construction we have shown that cover systems for weak bisimulation and trace equivalence arise through a distributive law for a monad. This work appears in [16].

We have shown that a certain subcategory of processes in SProc — the maps of SProc modulo trace equivalence — can be presented algebraicly as circuits, forming a potential basis for programming in process categories. This work is partially reported in [18].

1.5 Notation

We write $\_\times\_$ to indicate the product functor, with the projections p_0 and p_1 and diagonal Δ. The coproduct $\_+\_$ has injections b_0 and b_1, and codiagonal ∇. We write $\langle f, g\rangle$ to indicate the pairing $A \times B \to C$ of morphisms $f : A \to C$ and $g : B \to C$, and $\langle h|k\rangle$ as the copairing $C \to A + B$ of morphisms $h : C \to A$ and $k : C \to B$. The natural isomorphisms for associativity, symmetry and unit elimination of a monoidal category are written $a : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$, $c : A \otimes B \to B \otimes A$ and $u : \top \otimes A \to A$ respectively.

We take $\times$ to have higher binding precedence that $+$, both of which have higher precedence than the composition operator ;. We also adopt the convention that binary operators are left-associative, so that $A \otimes B \otimes C$ is interpreted as $(A \otimes B) \otimes C$.

A functor is *stable* if it preserves pullbacks, and a natural transformation is *cartesian* when all of its naturality squares are pullbacks. A category with coproducts is *extensive* if given the following commuting diagram

$$\begin{tikzcd}
X \arrow[r] \arrow[d] \arrow[dr, phantom, "(1)"] & Z \arrow[d] \arrow[dr, phantom, "(2)"] & Y \arrow[l] \arrow[d] \\
A \arrow[r, "b_0"'] & A+B & B \arrow[l, "b_1"]
\end{tikzcd}$$

(1) and (2) are pullbacks if and only if the top row is a pullback. Note that in an extensive category the coproduct functor is stable and both the injections and the codiagonal are cartesian. Note also that the injections are monic in an extensive category [12].

Chapter 2

Process categories

This chapter presents the construction of a category of processes upon a category of models of processes. The model category expresses the intended process dynamics, and provides a notion of behavioral equivalence in the form of a cover system. A process category is then formed as a span category quotiented by the cover system.

We begin with a brief discussion of model categories presented as sketches, followed by a detailed treatment of cover systems which is necessary for the subsequent development. We then present the process construction and show that it may be viewed as a 2-functor. This allows one to establish much of the structure of a process category at the level of it's model category. In particular, we consider how one may obtain compact-closure, biproducts and storage in a process category.

The techniques are illustrated using model categories of transition systems which give rise to Abramsky's category SProc. We conclude by considering other examples which arise by varying the parameters to the process construction.

2.1 Model categories

We will generally use the term *model category* to indicate any category which is amenable to the process construction of section 2.3. Typically, however, a model category for processes will be presented as a sketch (see Barr and Wells [7] or Wells [34]).

An example which will be used extensively for illustration is the category of deterministic labelled transition systems. These can be built upon any category $\mathbf{X}$ with finite limits:

Example 2.1.1 Tran($\mathbf{X}$) *is the category of models in* $\mathbf{X}$ *of the following sketch:*

$$\begin{tikzcd} & P \arrow[dl, "m"'] \arrow[dr, "\alpha"] & & \\ S\times\Sigma & & S & 1 \arrow[l, "i"'] \end{tikzcd}$$

Recall that for sketch $\mathcal{S}$, the model category $\mathcal{S}(\mathbf{X})$ has as objects diagrams in $\mathbf{X}$ and as morphisms "natural transformations". For example, a morphism $f : A \to B$ in Tran($\mathbf{X}$) corresponds to the following commuting diagram in $\mathbf{X}$:

$$\begin{tikzcd} S_A\times\Sigma_A \arrow[d, "f_S\times f_\Sigma"'] & P_A \arrow[l, "m_A"'] \arrow[r, "\alpha_A"] \arrow[d, "f_P"] & S_A \arrow[d, "f_S"] & 1 \arrow[l, "i_A"'] \arrow[d] \\ S_B\times\Sigma_B & P_B \arrow[l, "m_B"'] \arrow[r, "\alpha_B"] & S_B & 1 \arrow[l, "i_B"'] \end{tikzcd}$$

When $\mathbf{X}$ is the category **Set** of sets and functions, a transition system consists of a state set S with distinguished initial state $i \in S$, a label set Σ, and a partial function $S\times\Sigma \to S$ which specifies the allowable transitions between states. A morphism $f : A \to B$ is a pair (f_S, f_Σ) of functions which preserve initial states (i.e. $f(i_A) = i_B$) and preserve transitions (i.e. $(s, a) \in P_A$ implies $(fs, fa) \in P_B$ and $f(\alpha_A(s, a)) = \alpha_B(f(s), f(a))$).

Another category used for illustration is the category of trees [20], which may be constructed upon an arbitrary category $\mathbf{X}$:

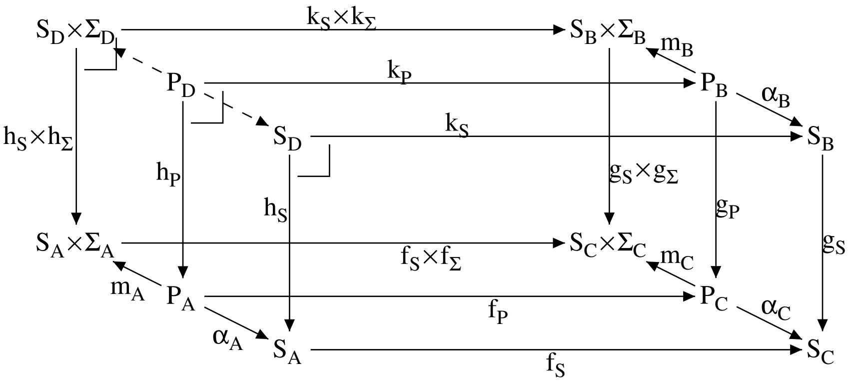
Example 2.1.2 Tree($\mathbf{X}$) *is the category of models in* $\mathbf{X}$ *of the following (infinite) sketch:*

$$\begin{tikzcd} 1 & S_0 \arrow[l, "f_0"'] & S_1 \arrow[l, "f_1"'] & S_2 \arrow[l, "f_2"'] & \cdots \arrow[l, "f_3"'] \end{tikzcd}$$

Of course a tree may be viewed as a transition system in which all transitions are uniquely labelled and all states are reachable by exactly one sequence of transitions: S_0 are the states directly reachable from the initial state, and $f_0^{-1}(s)$ are the states directly reachable from state $s \in S_0$.

We will see later that the process construction requires a model category to have certain limits and that certain functorial structure in a model category induces related structure in it's process category. For the sketches considered in this thesis, the associated model categories will have finite limits which are given pointwise. In Tran($\mathbf{X}$), for example, the pullback of morphisms $f : A \to C$

and $g : B \to C$ (omitting initial states) is given as follows:



We will also be interested in model categories which have extensive coproducts. These too will be given pointwise when they exist, as is the case in the category of trees. One obtains a coproduct of transition systems by coallescing initial states, but since initial states are reachable the resulting coproduct is not disjoint and thus not extensive.

2.2 Cover systems

Cover systems are central to the subsequent construction of process categories. This section examines the basic properties of cover systems and provide various techniques for constructing cover systems — in particular, upon the model categories which arise from sketches.

Definition 2.2.1 *A class $\mathcal{X}$ of morphisms in a category $\mathbf{X}$ is a* cover system *provided that: $\mathcal{X}$ contains all isomorphisms; $\mathcal{X}$ is closed under composition; and for x in $\mathcal{X}$ and f in $\mathbf{X}$, the pullback*

$$\begin{tikzcd} \cdot \arrow[r] \arrow[d, "z"'] \arrow[dr, phantom, "\lrcorner", very near start] & \cdot \arrow[d, "x"] \\ \cdot \arrow[r] & \cdot \end{tikzcd}$$

exists with z in $\mathcal{X}$.

We say that a cover system $\mathcal{X}$ is *left-factor closed* if the fact that $f;g$ and g are in $\mathcal{X}$ implies that f is in $\mathcal{X}$. Similarly, $\mathcal{X}$ is *right-factor closed* if the fact that $f;g$ and f are in $\mathcal{X}$ implies g is in $\mathcal{X}$.

There are many commonly occuring examples of cover systems: The class $\mathcal{I}$ of isomorphisms in any category is a left-factor closed cover system. In category $\mathbf{X}$ with pullbacks the classes $\mathbf{X}_1$

of all morphisms and $\mathcal{M}$ of monics are each left-factor closed cover systems, while the class $\mathcal{R}$ of retractions is a right-factor closed cover system. In a regular category, the regular epimorphisms form a cover system which is right-factor closed. If $\mathcal{X}_1$ and $\mathcal{X}_2$ are cover systems, the intersection $\mathcal{X}_1 \cap \mathcal{X}_2$ is a cover system which is left- (resp. right) factor closed whenever both $\mathcal{X}_1$ or $\mathcal{X}_2$ are left- (resp. right) factor closed.

A fact that plays a rather technical role in the subsequent development is that any cover system can be closed by adding all of its right factors, thus obtaining a cover system which is right-factor closed:

$$\mathcal{X}{\downarrow} \stackrel{def}{=} \{\, z \in \mathbf{X}_1 \mid \exists\, x \in \mathcal{X}.\ x; z \in \mathcal{X} \,\}$$

Of course if $\mathcal{X}$ is already right-factor closed, then $\mathcal{X}{\downarrow}$ is simply $\mathcal{X}$.

Lemma 2.2.2 *$\mathcal{X}{\downarrow}$ is a right-factor closed cover system containing $\mathcal{X}$, which inherits left-factor closure from $\mathcal{X}$.*

Proof. We will say that x is a witness that $z \in \mathcal{X}{\downarrow}$ when both x and $x; z$ are in $\mathcal{X}$. Note that $\mathcal{X}{\downarrow}$ contains $\mathcal{X}$ (and thus all isomorphisms) as witnessed by the identity morphisms. To see compositionality, suppose x and x' are witness that z and z' belong to $\mathcal{X}{\downarrow}$:

$$\begin{tikzcd}
 & & \cdot \arrow[dr, "z"] & & \\
 & \cdot \arrow[ur, "x"] \arrow[rr] & & \cdot \arrow[dr, "z'"] & \\
\cdot \arrow[ur, "x''"] \arrow[rr] & & \cdot \arrow[ur, "x'"] \arrow[rr] & & \cdot
\end{tikzcd}$$

then $x''; x$ is witness that $z; z'$ is in $\mathcal{X}{\downarrow}$. To see stability, suppose x is witness that z is in $\mathcal{X}{\downarrow}$. Forming the pullback along an arbitrary f

$$\begin{tikzcd}
\cdot \arrow[r, "x'"] \arrow[d] \arrow[dr, phantom, very near start, "\lrcorner"] & \cdot \arrow[r, "z'"] \arrow[d] \arrow[dr, phantom, very near start, "\lrcorner"] & \cdot \arrow[d, "f"] \\
\cdot \arrow[r, "x"'] & \cdot \arrow[r, "z"'] & \cdot
\end{tikzcd}$$

yields the required witness x' for z'.

To see $\mathcal{X}{\downarrow}$ inherits left-factor closure, let z and $f; z$ belong to $\mathcal{X}{\downarrow}$ as witnessed by x and y respectively and consider the following diagram:

$$\begin{tikzcd}
\cdot \arrow[r, "y'"] \arrow[d, "x''"'] \arrow[dr, phantom, very near start, "\lrcorner"] & \cdot \arrow[r, "f'"] \arrow[d, "x'"'] \arrow[dr, phantom, very near start, "\lrcorner"] & \cdot \arrow[d, "x"] & \\
\cdot \arrow[r, "y"'] & \cdot \arrow[r, "f"'] & \cdot \arrow[r, "z"'] & \cdot
\end{tikzcd}$$

As $x''; y; f; z$ is in $\mathcal{X}$, we have $y'; f'$ in $\mathcal{X}$ by left-factor closure and consequently f' is in $\mathcal{X}{\downarrow}$. Then f is in $\mathcal{X}{\downarrow}$ by right-factor closure. $\square$

We note the following result concerning preservation of cover systems by functors, where we suppose $\mathbf{X}$ and $\mathbf{Y}$ are categories with cover systems $\mathcal{X}$ and $\mathcal{Y}$ respectively:

Lemma 2.2.3 *If $F(\mathcal{X}) \subseteq \mathcal{Y}{\downarrow}$ then $F(\mathcal{X}{\downarrow}) \subseteq \mathcal{Y}{\downarrow}$.*

Proof. Suppose x is witness that r is in $\mathcal{X}{\downarrow}$, and that y_1 and y_2 are witness that $F(x; r)$ and Fx are in $\mathcal{Y}{\downarrow}$ respectively. Forming the pullback

$$\begin{tikzcd}
\cdot \arrow[r] \arrow[d, "y_3"'] \arrow[dr, phantom, very near start, "\lrcorner"] & \cdot \arrow[d, "y_2"] \arrow[dr, dashed] & & \\
\cdot \arrow[r, "y_1"'] & \cdot \arrow[r, "Fx"'] & \cdot \arrow[r, "Fr"'] & \cdot
\end{tikzcd}$$

gives the witness $y_3; y_1; Fx$ that Fr is in $\mathcal{Y}{\downarrow}$. $\square$

Some important techniques for obtaining cover systems derive from the following notion and its associated properties: A commuting square in $\mathbf{X}$ is an $\mathcal{X}$*-pullback* if the induced morphism to the inscribed pullback is in $\mathcal{X}$.

$$\begin{tikzcd}
\cdot \arrow[rr] \arrow[dd] \arrow[dr, dashed, "x"] & & \cdot \arrow[dd] \\
& \cdot \arrow[ur] \arrow[dl] \arrow[dr, phantom, very near start, "\lrcorner"] & \\
\cdot \arrow[rr] & & \cdot
\end{tikzcd}$$

Clearly, f belongs to $\mathcal{X}$ if and only if the square $f; 1 = f; 1$ is an $\mathcal{X}$-pullback.

Lemma 2.2.4 *Consider the following commuting diagram in $\mathbf{X}$:*

$$\begin{tikzcd}
\cdot \arrow[r, "u"] \arrow[d] \arrow[dr, phantom, "(1)"] & \cdot \arrow[r] \arrow[d] \arrow[dr, phantom, "(2)"] & \cdot \arrow[d] \\
\cdot \arrow[r, "v"'] & \cdot \arrow[r] & \cdot
\end{tikzcd}$$

i) the outer square is an $\mathcal{X}$-pullback whenever (1) and (2) are $\mathcal{X}$-pullbacks;

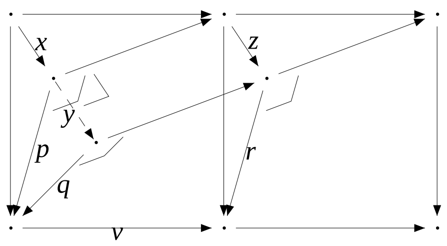
ii) (1) is an $\mathcal{X}$-pullback whenever (2) is a pullback and the outer square is an $\mathcal{X}$-pullback;

iii) if $\mathcal{X}$ is left-factor closed then (1) is an $\mathcal{X}$ pullback whenever the outer square and (2) are $\mathcal{X}$-pullbacks;

iv) if (1) is an $\mathcal{X}$-pullback whenever the outer square and (2) are $\mathcal{X}$-pullbacks, then $\mathcal{X}$ is left-factor closed;

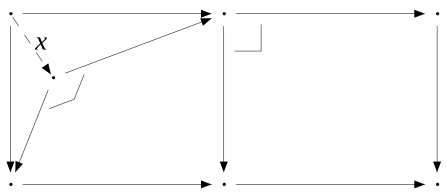
v) if the outer square is an $\mathcal{X}{\downarrow}$-pullback and u and v belong to $\mathcal{X}{\downarrow}$ then (2) is an $\mathcal{X}{\downarrow}$-pullback.

Proof. i) Let x and z witness that (1) and (2) are $\mathcal{X}$-pullbacks.

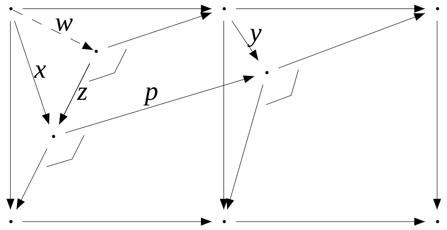


Forming the pullback of r and v, one obtains the pullback inscribed in the entire square; the induced morphism to this pullback is $x;y$. As y is a pullback of z it belongs to $\mathcal{X}$ and thus $x;y$ belongs to $\mathcal{X}$ also.

ii) The induced morphism to the pullback inscribed in (1) is also the induced morphism to the pullback inscribed in the outer square.

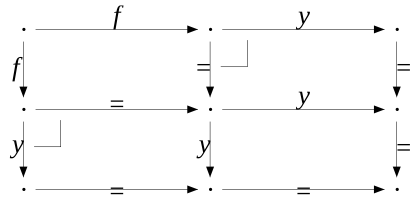


iii) Suppose $\mathcal{X}$ is left-factor closed and let x and y witness that the outer square and (2) are $\mathcal{X}$-pullbacks.



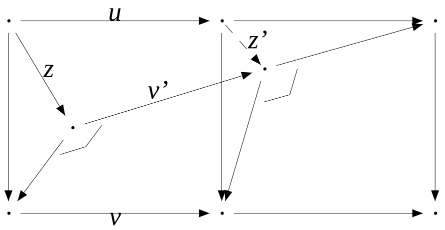
Forming the pullback of y along p gives a pullback inscribed in (1). The induced morphism w then belongs to $\mathcal{X}$ since x and z belong to $\mathcal{X}$.

iv) Suppose $f;y$ and y are in $\mathcal{X}$. Using parts (i) and (ii) above, the top left square of



is an $\mathcal{X}$-pullback and thus f is in $\mathcal{X}$.

v) In the following diagram



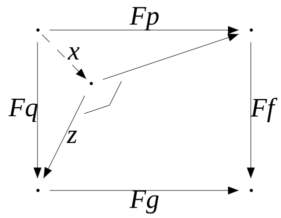
z' is a right-factor $z;v'$ and thus belongs to $\mathcal{X}{\downarrow}$.

$\square$

The following lemma shows that certain functors induce a cover system in their domain category.

Lemma 2.2.5 *If $F : \mathbf{Y} \to \mathbf{X}$ takes pullbacks to $\mathcal{X}$-pullbacks then $F^{-1}(\mathcal{X})$ is a cover system on $\mathbf{Y}$ which inherits left-factor and right-factor closure from $\mathcal{X}$.*

Proof. As functors preserve isomorphisms and composition, the proposed cover system contains all isomorphisms and is closed to composition. To see stability, suppose Ff is in $\mathcal{X}$ and that $p;f = q;g$ is a pullback in $\mathbf{Y}$. Then in the diagram



both x and z are in $\mathcal{X}$ and thus Fq is in $\mathcal{X}$ as required.
That $F^{-1}(\mathcal{X})$ is left-factor (resp. right-factor) closed whenever $\mathcal{X}$ is left-factor (resp. right-factor) closed is a simple consequence of the fact that F preserves composition. $\square$

As a simple example, note that $\mathrm{Tran}(\mathbf{X})$ is fibred over $\mathbf{X}$ and that the fibration functor (which takes a transition system to its alphabet) is stable. One obtains a cover system in $\mathrm{Tran}(\mathbf{X})$ consisting of all morphisms whose label component is an isomorphism.

We turn now to obtaining cover systems upon model categories which arise from sketches. Let F and G be functors $\mathbf{Y} \to \mathbf{X}$, and f any morphism of $\mathbf{X}$. We say that a natural transformation $F \Rightarrow G$ is *$\mathcal{X}$-cartesian for f* if the naturality square associated with f is an $\mathcal{X}$-pullback.

Proposition 2.2.6 *The natural transformations $F \Rightarrow G$ which are $\mathcal{X}$-cartesian for f form a cover system on* $\mathrm{Func}(\mathbf{Y}, \mathbf{X})$*; this cover system is left-factor closed iff $\mathcal{X}$ is left-factor closed.*

Proof. The proposed cover system contains natural isomorphisms, as these are cartesian, and is closed to composition by part (i) of lemma 2.2.4. To see that it is closed to pullback, suppose α is $\mathcal{X}$-cartesian for y and that γ results from pulling back α along arbitrary β:

$$\begin{tikzcd}
\cdot \arrow[rr, "\gamma_A"] \arrow[dr] \arrow[dd, dashed, "Py"'] & & \cdot \arrow[dr, "\beta_A"] \arrow[dd, "Hy"' near start] & \\
& \cdot \arrow[rr, "\alpha_A" near start] \arrow[dd, "Fy"' near start] & & \cdot \arrow[dd, "Gy"] \\
\cdot \arrow[rr, "\gamma_B" near start] \arrow[dr] & & \cdot \arrow[dr, "\beta_B"] & \\
& \cdot \arrow[rr, "\alpha_B"'] & & \cdot
\end{tikzcd}$$

As the front face above is an $\mathcal{X}$-pullback, the rear face is also by parts (i) and (ii) of lemma 2.2.4. Thus γ is $\mathcal{X}$-cartesian for y.

The result concerning left-factor closure is given directly by parts (iii) and (iv) of lemma 2.2.4. □

As the category of models of a sketch $\mathcal{S}$ is essentially such a functor category, we can obtain a cover system by choosing an arrow σ (or by intersection, any number of arrows) of $\mathcal{S}$:

Corollary 2.2.7 *The class of morphisms of $\mathcal{S}(\mathbf{X})$ which are $\mathcal{X}$-cartesian for σ form a cover system; this cover system is left-factor closed iff $\mathcal{X}$ is left-factor closed.*

In this way one obtains various useful cover systems upon categories of transition systems of example 2.1.1. Let $\mathcal{X}$ be a cover system on $\mathbf{X}$:

Example 2.2.8 *In* $\mathrm{Tran}(\mathbf{X})$*, define $\exists^{\mathcal{X}}$ to be the class of morphisms which are $\mathcal{X}$-cartesian for the arrow $m; p_0$.*

With respect to transitions, the morphisms of $\exists^{\mathcal{X}}$ are *locally* $\mathcal{X}$ in the following sense:

$$
\begin{tikzcd}
P^s \arrow[rr] \arrow[dr, dashed, "x'"] \arrow[dd] & & P_A \arrow[rr, "f_P"] \arrow[dr, "x"] \arrow[dd] & & P_B \arrow[dd, "m_B;p_0"] \\
& P_{f(s)} \arrow[rr] \arrow[dl] & & \cdot \arrow[ur] \arrow[dl] & \\
1 \arrow[rr, "s"'] & & S_A \arrow[rr, "f_S"'] & & S_B
\end{tikzcd}
$$

if $f : A \to B \in \exists^{\mathcal{X}}$ then at any state s of A, the transitions from s (viz. P^s) map via a morphism of $\mathcal{X}$ to the transitions from $f(s)$. In a regular category, for example, the morphisms of $\exists^{\mathcal{E}}$ are local epimorphisms: at each state s of A, every transition from state $f(s)$ in B has at least one corresponding transition from s in A.

The cover system $\exists^{\mathcal{E}}$ of morphisms which "reflect" transitions can be isolated in a similar way in any related category of transition systems, and have a history of use in describing bisimulation (see [21]). We will use $\exists^{\mathcal{E}}$ to give bisimulation of processes in SProc, but the classes $\exists^{\mathcal{I}}$ of local isomorphisms and $\exists^{\mathcal{M}}$ of local monics will also be important in the subsequent development.

Recall the sketch of trees in example 2.1.2:

Example 2.2.9 *In* Tree($\mathbf{X}$)*, define* $\exists^{\mathcal{X}}$ *to be the class of morphisms which are* $\mathcal{X}$*-cartsian for every arrow* f_i*.*

In Tree($\mathbf{X}$), due to the initial state, any morphism of $\exists^{\mathcal{X}}$ has all of it's components in $\mathcal{X}$. For example, any morphism in $\exists^{\mathcal{E}}$ is itself an epimorphism. Note also that any monic of Tree($\mathbf{X}$) is componentwise monic in $\mathbf{X}$ and thus belongs to $\exists^{\mathcal{M}}$ by left-factor closure.

Although trees are unlabelled, one can view a morphism of trees as introducing labels: given $f : A \to B$, "transitions" $a \in A_i$ and $b \in B_i$ have the same label when $f_i(a_i) = b_i$. We can thus view the regular epimorphisms as providing a cover system for trace equivalence of trees.

2.3 The process construction

A process category is built upon a model category through a generalized span construction: span legs are taken from a pair of cover systems, and span equivalence is dictated by a third cover system on the model category. This construction may be viewed as a 2-functor, a fact which is useful for establishing the structure of a process category in terms of the structure of it's model category.

2.3.1 The category of processes

Let $\mathbf{X}$ be a category with cover systems $\mathcal{X}_0$ and $\mathcal{X}_1$. One forms the bicategory $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$ of spans in $\mathbf{X}$ as follows:

- the objects A are those of $\mathbf{X}$
- the hom-category $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})(A,B)$ has as objects $f : A \rightsquigarrow B$ those spans

$$\begin{tikzcd} & \cdot \arrow[dl, "f_0"'] \arrow[dr, "f_1"] & \\ A & & B \end{tikzcd}$$

  in $\mathbf{X}$ with f_0 in $\mathcal{X}_0$ and f_1 in $\mathcal{X}_1$, and as arrows $f \Rightarrow g$ the morphisms x of $\mathbf{X}$ for which

$$\begin{tikzcd} & \cdot \arrow[dl, "f_0"'] \arrow[dr, "f_1"] \arrow[dd, "x"] & \\ A & & B \\ & \cdot \arrow[ul, "g_0"] \arrow[ur, "g_1"'] & \end{tikzcd}$$

- the horizontal identity on A is given by a span $(1_A, 1_A)$ of identities in $\mathbf{X}$, and horizontal composition is given by pullback — i.e. the composite $f \mathbin{;;} g$ of 1-cells $f : A \rightsquigarrow B$ and $g : B \rightsquigarrow C$ below

$$\begin{tikzcd} & & & & \cdot \arrow[dlll, "h_0"'] \arrow[drrr, "h_1"] \arrow[dd, dashed, "z"] & & & & \\ & \cdot \arrow[dddl, "f_0"'] \arrow[dddrrr, "f_1"] \arrow[ddr, "x"] & & & & & & \cdot \arrow[dddlll, "g_0"'] \arrow[dddr, "g_1"] \arrow[ddl, "y"'] & \\ & & & & \cdot \arrow[dll] \arrow[drr] & & & & \\ & & \cdot \arrow[dll] \arrow[drr] & & & & \cdot \arrow[dll] \arrow[drr] & & \\ A & & & & B & & & & C \end{tikzcd}$$

  is the span $(h_0; f_0, h_1; g_1)$, and the composite $x \mathbin{;;} y$ of 2-cells $x : f \Rightarrow f'$ and $y : g \Rightarrow g'$ is the induced 2-cell $z : f \mathbin{;;} g \Rightarrow f' \mathbin{;;} g'$ above.

We will adopt the convention that omitted cover system parameters are implicitly specified as the class of all morphisms in the base category. Thus, for example, $\mathrm{Span}(\mathbf{X})$ is the standard bicategory of spans in $\mathbf{X}$ [8].

Proposition 2.3.1 $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$ *is a bicategory.*

Proof. As we are simply restricting the standard bicategory of spans, it is enough to note that the cover system axioms ensure the restricted hom-categories remain categories. $\square$

Of course the indended view of a span

$$
\begin{tikzcd}
& P \arrow[dl, "f"'] \arrow[dr, "g"] & \\
A & & B
\end{tikzcd}
$$

is as a process with interface specifications A and B which abstract the implementation P. The use of cover systems for span legs provides a compositional means to impose constraints on how a process implements its interfaces. Span composition by pullback enacts process interaction at the shared interface.

Consider a span of arbitrary morphisms in the category of transition systems. As an interface specification, a transition system indicates what interaction is permitted at each state and how interaction affects the state of the interface. Note that although the transition systems themselves are deterministic, the emergent processes are nondeterministic since a transition in the interface may be implemented by several alternative transitions in the span apex. Composing processes by pullback yields a composite process whose states are the pairs of component states and whose transitions are those pairs of component transitions which synchronize at the shared interface.

Turning now to process equivalence. We note from Pare [28] that given a bicategory $\mathcal{B}$ and sub-bicategory $\mathcal{B}'$, one can turn the 1-cell structure of $\mathcal{B}$ into a category by quotienting with the equivalence relation generated by the 2-cells of $\mathcal{B}'$.

The cover system axioms ensure that any cover system $\mathcal{X}$ on $\mathbf{X}$ serves as the 2-cells of a "wide" sub-bicategory of $\mathrm{Span}(\mathbf{X})$ whose hom-categories have pullbacks. The induced equivalence relation, say $\sim_{\mathcal{X}}$, equates spans f and g exactly when there exist a span (x, x') of 2-cells belonging to $\mathcal{X}$:

$$
\begin{tikzcd}
& \cdot \arrow[dl, "f_0"'] \arrow[dr, "f_1"] & \\
A & \cdot \arrow[l, "h_0"] \arrow[r, "h_1"'] \arrow[u, "x"'] \arrow[d, "x'"] & B \\
& \cdot \arrow[ul, "g_0"] \arrow[ur, "g_1"'] &
\end{tikzcd}
$$

In this way we obtain the category of processes in $\mathbf{X}$:

Definition 2.3.2 $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X}, \mathcal{X})$ *is the category which results from quotienting the bicategory* $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$ *with the cover system* $\mathcal{X}$.

Note that if $\mathcal{X}_0$ (resp. $\mathcal{X}_1$) is left-factor closed then the $\mathcal{X}$ is effectively constrained to lie within $\mathcal{X}_0$ (resp $\mathcal{X}_1$). We will use the convention that if the parameter $\mathcal{X}$ is unspecified then it is taken to be the intersection of $\mathcal{X}_0$ and $\mathcal{X}_1$.

Well known examples of process categories are categories of relations and partial maps: for regular category $\mathbf{E}$ with regular epimorphisms $\mathcal{E}$, the process category $\mathrm{Proc}(\mathbf{E}, \mathcal{E})$ is the category of relations in $\mathbf{E}$; for $\mathbf{X}$ a category with pullbacks, the process category $\mathrm{Proc}^{\mathcal{M}}(\mathbf{X}, \mathcal{I})$ is the category of partial maps in $\mathbf{X}$.

If one takes the model category to be transition systems upon $\mathbf{Set}$ and takes the cover system for equivalence to be the class $\exists^{\mathcal{E}}$ of local epimorphisms, the resulting process category is equivalent to Abramsky's category SProc of synchronous processes modulo bisimulation:

$$\mathrm{SProc} \;\equiv\; \mathrm{Proc}(\mathrm{Tran}(\mathbf{Set}), \exists^{\mathcal{E}})$$

Surprisingly, one also obtains an equivalent process category by taking the model category of trees and the cover system for bisimulation there:

$$\mathrm{SProc} \;\equiv\; \mathrm{Proc}(\mathrm{Tree}(\mathbf{Set}), \exists^{\mathcal{E}})$$

These statements were justified in [17] and will be revisited in chapter 4.

Recalling that trace equivalence of trees is given by the regular epimorphisms, we make the following definition:

Definition 2.3.3 *The category* SProc_T *of synchronous processes modulo trace equivalence is the category* $\mathrm{Proc}(\mathrm{Tree}(\mathbf{Set}), \mathcal{E})$ *of relations on the category of trees.*

2.3.2 Lifting functors and natural transformations

Given a functor $F : \mathbf{X} \to \mathbf{Y}$ between model categories, one can ask when the following operation on spans

$$\begin{tikzcd} & P \arrow[dl, "h_0"'] \arrow[dr, "h_1"] & \\ A & & B \end{tikzcd} \quad\longmapsto\quad \begin{tikzcd} & FP \arrow[dl, "Fh_0"'] \arrow[dr, "Fh_1"] & \\ FA & & FB \end{tikzcd}$$

gives a functor $\mathrm{Proc}(F)$ between the associated process categories.

Lemma 2.3.4 $\mathrm{Proc}(F)$ *is a functor* $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X}, \mathcal{X}) \to \mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y}, \mathcal{Y})$ *if and only if :*

(1) $F(\mathcal{X}_0) \subseteq \mathcal{Y}_0$ *and* $F(\mathcal{X}_1) \subseteq \mathcal{Y}_1$,

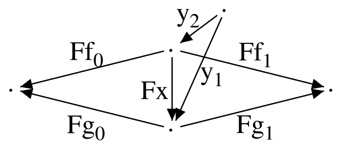
(2) $F(x) \in \mathcal{Y}{\downarrow}$ *whenever* $x \in \mathcal{X}$ *and* x *is a 2-cell in* $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$,

(3) and $F(r_0); F(f_1) = F(r_1); F(g_0)$ is a $\mathcal{Y}{\downarrow}$-pullback whenever

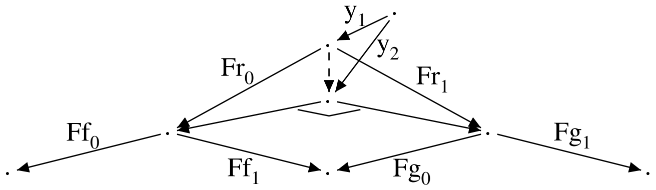
$$\begin{tikzcd} \cdot \arrow[r, "r_1"] \arrow[d, "r_0"'] \arrow[dr, phantom, "\lrcorner", very near start] & \cdot \arrow[d, "g_0"] \\ \cdot \arrow[r, "f_1"'] & \cdot \end{tikzcd}$$

is a pullback with r_0 and g_0 in $\mathcal{X}_0$ and r_1 and f_1 in $\mathcal{X}_1$.

Proof. For $\mathrm{Proc}(F)$ to preserve equality of processes it is necessary and sufficient that any witness of equality in $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$ is taken to a right-factor of a withness of equality in $\mathrm{Span}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y})$:



Regarding composition, there is always a 2-cell $\mathrm{Proc}(F)(f;g) \to Proc(F)(f); \mathrm{Proc}(F)(g)$ in $\mathrm{Span}(\mathbf{Y})$. For $\mathrm{Proc}(F)$ to preserve composition it is then necessary and sufficient that this 2-cell belong to $\mathcal{Y}{\downarrow}$.



Finally, $\mathrm{Proc}(F)$ preserves identities since F is a functor. □

For most of the process categories we will consider, the legs of spans are not restricted. In this case we write $F : (\mathbf{X}, \mathcal{X}) \to (\mathbf{Y}, \mathcal{Y})$ to indicate that $\mathrm{Proc}(F)$ is a functor, and we note the following alternate characterization:

Lemma 2.3.5 *$F : (\mathbf{X}, \mathcal{X}) \to (\mathbf{Y}, \mathcal{Y})$ if and only if F takes $\mathcal{X}$-pullbacks to $\mathcal{Y}{\downarrow}$-pullbacks.*

Proof. Suppose F takes $\mathcal{X}$-pullbacks to $\mathcal{Y}{\downarrow}$-pullbacks. Then the $\mathcal{X}$-pullback $x; 1 = x; 1$ is taken to the $\mathcal{Y}{\downarrow}$-pullback $Fx; 1 = Fx; 1$, so $Fx \in \mathcal{Y}$. Conversely, if x is witness that $h; f = k; g$ is an

$\mathcal{X}$-pullback then $Fx; y$ is witness that the outer square below

$$
\begin{tikzcd}
\cdot \arrow[rrr, "Fh"] \arrow[dr, dashed, "Fx"] \arrow[ddd, "Fk"'] & & & \cdot \arrow[ddd, "Ff"] \\
& \cdot \arrow[urr] \arrow[ddl] \arrow[dr, dashed, "y"] & & \\
& & \cdot \arrow[uur] \arrow[ddll] & \\
\cdot \arrow[rrr, "Fg"'] & & & \cdot
\end{tikzcd}
$$

is a $\mathcal{Y}\!\downarrow$-pullback. $\square$

We note another special case when the cover systems for span legs are left-factor closed and the cover system for equivalence is their intersection:

Lemma 2.3.6 *If $\mathcal{Y}_0$ and $\mathcal{Y}_1$ are left-factor closed then* $\mathrm{Proc}(F)$ *is a functor* $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X},\mathcal{X}) \to \mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y})$ *provided* $F(\mathcal{X}_0) \subseteq \mathcal{Y}_0$, $F(\mathcal{X}_1) \subseteq \mathcal{Y}_1$, *and* $F(\mathcal{X}) \subseteq \mathcal{Y}_0 \cap \mathcal{Y}_1$.

Proof. Suppose $r_0; f_1 = r_1; g_0$ is a pullback with r_0 and g_0 in $\mathcal{X}_0$ and r_1 and f_1 in $\mathcal{X}_1$. Then the induced morphism in the following diagram

$$
\begin{tikzcd}
\cdot \arrow[rr, "Fr_0"] \arrow[dr, dashed] \arrow[dd, "Fr_1"'] & & \cdot \arrow[dd, "Ff_1"] \\
& \cdot \arrow[ur, "y_0"] \arrow[dl, "y_1"] & \\
\cdot \arrow[rr, "Fg_0"'] & & \cdot
\end{tikzcd}
$$

is in $\mathcal{Y}_0 \cap \mathcal{Y}_1$ by left-factor closure. $\square$

We now ask when a natural transformation $\alpha : F \Rightarrow G : \mathbf{X} \to \mathbf{Y}$ induces a natural transformation $\mathrm{Proc}(\alpha)$ between lifted functors:

$$
\begin{tikzcd}
& FA \arrow[dl, "="'] \arrow[dr, "\alpha_A"] & \\
FA & & GA
\end{tikzcd}
$$

Lemma 2.3.7 $\mathrm{Proc}(\alpha)$ *is a natural transformation* $\mathrm{Proc}(F) \Rightarrow \mathrm{Proc}(G) : \mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X},\mathcal{X}) \to \mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y},\mathcal{Y})$ *if and only if :*

(1) α belongs componentwise to $\mathcal{X}_1$

(2) α is $\mathcal{Y}\!\downarrow$-cartesian for the morphisms of $\mathcal{X}_0$.

Proof. A naturality square $\mathrm{Proc}(F)(f);\mathrm{Proc}(\alpha) = \mathrm{Proc}(\alpha);\mathrm{Proc}(G)(f)$ appears as follows in $\mathbf{Y}$:

$$\begin{tikzcd}
FA & & FA \arrow[ll, "="'] \arrow[rr, "\alpha_A"] & & GA \\
 & & & \cdot \arrow[ul] \arrow[dr] & \\
FP \arrow[uu, "Ff_0"] \arrow[dd, "Ff_1"'] & & FP \arrow[ll, "="] \arrow[uu] \arrow[dd] \arrow[ur, dashed, "y"] \arrow[rr, "\alpha_P"'] & & GP \arrow[uu, "Gf_0"'] \arrow[dd, "Gf_1"] \\
 & & & & \\
FB & & FB \arrow[ll, "="] \arrow[rr, "\alpha_B"'] & & GB
\end{tikzcd}$$

To commute in the process category means that the induced morphism y belongs to $\mathcal{Y}{\downarrow}$. $\square$

We note again a special case for left-factor closure:

Lemma 2.3.8 *If $\mathcal{X}$ is left-factor closed, then $\alpha : F \Rightarrow G$ is $\mathcal{X}$-cartesian whenever α_A is in $\mathcal{X}$ for all A in $\mathbf{Y}$.*

Proof. Consider the naturality square associated with any $y : A \to B$:

$$\begin{tikzcd}
FA \arrow[rr, "\alpha_A"] \arrow[dr, dashed, "z"] \arrow[dd, "Fy"'] & & GA \arrow[dd, "Gy"] \\
 & \cdot \arrow[ur] \arrow[dl] & \\
FB \arrow[rr, "\alpha_B"'] & & GB
\end{tikzcd}$$

The arrow opposite α_B in the pullback square is in $\mathcal{X}$ and thus the induced arrow z is in $\mathcal{X}$ by left-factor closure. $\square$

The following proves useful for identifying functorial structure which lies in the domain of the process construction.

Lemma 2.3.9 *Let $\phi : F \Rightarrow G : \mathbf{X} \to \mathbf{Y}$ be $\mathcal{Y}{\downarrow}$-cartesian and pointwise in $\mathcal{Y}{\downarrow}$.*

(1) If $F : (\mathbf{X}, \mathcal{X}) \to (\mathbf{Y}, \mathcal{Y})$ then $G : (\mathbf{X}, \mathcal{X}) \to (\mathbf{Y}, \mathcal{Y})$.

(2) If $\alpha : F \Rightarrow H$ and $\beta : H \Rightarrow K$ are $\mathcal{Y}{\downarrow}$-cartesian and $\alpha;\beta = \phi;\gamma$, then γ is $\mathcal{Y}{\downarrow}$-cartesian.

Proof. Note that if the top face of the following diagram is an $\mathcal{Y}{\downarrow}$-pullback

$$\begin{tikzcd}
 & FA \arrow[rr] \arrow[dd, "\phi"] & & FC \arrow[dd, "\phi"] \\
FP \arrow[ur] \arrow[rr] \arrow[dd, "\phi"'] & & FB \arrow[ur] \arrow[dd, "\phi"] & \\
 & GA \arrow[rr] & & GC \\
GP \arrow[ur] \arrow[rr] & & GB \arrow[ur] &
\end{tikzcd}$$

then, since ϕ_P and ϕ_A are in $\mathcal{Y}{\downarrow}$, the bottom face is also a $\mathcal{Y}{\downarrow}$-pullback by lemma 2.2.4. Part (2) follows directly from the same lemma. $\square$

2.3.3 The 2-functor Proc

We can thus view the process construction as a 2-functor, with domain the 2-category $\underline{\mathbf{PModel}}$ of process models given by

- 0-cells – tuples $(\mathbf{X}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_1)$, where $\mathbf{X}$ is a category with cover systems $\mathcal{X}_0$, $\mathcal{X}_1$ and $\mathcal{X}$,
- 1-cells – functors $F : \mathbf{X} \to \mathbf{Y}$ which satisfy lemma 2.3.4,
- 2-cells – natural transformations $\alpha : F \Rightarrow G$ which satisfy lemma 2.3.7.

Proposition 2.3.10 $\mathrm{Proc} : \underline{\mathbf{PModel}} \to \underline{\mathbf{Cat}}$ *is a 2-functor which preserves products.*

Proof. It is clear that Proc preserves 1-cell composition (i.e. $\mathrm{Proc}(F\,;;\,G)$ is $\mathrm{Proc}(F)\,;;\,\mathrm{Proc}(G)$) and vertical 2-cell composition (i.e. $\mathrm{Proc}(\alpha); \mathrm{Proc}(\beta)$ is $\mathrm{Proc}(\alpha; \beta)$). To see that horizontal composition of 2-cells is preserved, suppose $\alpha : F \Rightarrow F'$ and $\beta : G \Rightarrow G'$. Then

$$\begin{aligned}
\mathrm{Proc}(\alpha\,;;\,\beta) &\equiv \mathrm{Proc}(\beta; G'\alpha) \\
&= \mathrm{Proc}(\beta); \mathrm{Proc}(G'\alpha) \\
&= \mathrm{Proc}(\beta); \mathrm{Proc}(G')(\mathrm{Proc}(\alpha)) \\
&\equiv \mathrm{Proc}(\alpha)\,;;\,\mathrm{Proc}(\beta)
\end{aligned}$$

as required.

It is easily seen that $\mathrm{Span}(\mathbf{X}) \times \mathrm{Span}(\mathbf{Y})$ is equal to $\mathrm{Span}(\mathbf{X} \times \mathbf{Y})$, giving preservation of products.
□

2.4 Linear process categories

One of the tenets in our approach to constructing process categories is that the functorial structure on the process category can often be predicted from related structure in the model category. This chapter examines how this applies to the elements of a *linear* process category.

2.4.1 Compact-closure

Products in any category preserve pullbacks and also preserve any cover system since the following squares are always pullbacks:

$$\begin{tikzcd}
A\times C \arrow[r, "f\times 1"] \arrow[d, "p_0"'] \arrow[dr, phantom, "\lrcorner", very near start] & B\times C \arrow[d, "p_0"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}
\qquad
\begin{tikzcd}
B\times C \arrow[r, "1\times g"] \arrow[d, "p_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & B\times D \arrow[d, "p_1"] \\
C \arrow[r, "g"'] & D
\end{tikzcd}$$

As isomorphisms, the associativity, symmetry and unit elimination transformations are cartesian and thus the product in any model category induces a symmetric tensor on it's process category.

Given a covariant involution $\_^{\perp}$ in a model category $\mathbf{X}$, one obtains a contravariant involution $(f, g) \mapsto (g^{\perp}, f^{\perp})$ on the process category $\mathrm{Proc}(\mathbf{X}, \mathcal{X})$. If the involution on $\mathbf{X}$ is such that $(A^{\perp}\times B^{\perp})^{\perp} = A\times B$, then $\mathrm{Proc}(\mathbf{X}, \mathcal{X})$ is immediately linearly distributive with the linear distribution δ given by associativity [14]. To further obtain compact closure it is sufficient to have, for each A, a span τ_A

$$\begin{tikzcd} & FA \arrow[dl, "!"'] \arrow[dr, "\gamma"] & \\ 1 & & A\times A^{\perp} \end{tikzcd}$$

and a morphism x of $\mathcal{X}$ for which:

$$\begin{tikzcd}
 & & & 1\times A \arrow[rrrddd, bend left, "c"] & & & \\
 & & & \cdot \arrow[u, "x"] \arrow[dll] \arrow[drr] & & & \\
 & FA\times A \arrow[dl, "!\times 1"'] \arrow[dr, "\gamma\times 1"] & & & & A\times FA^{\perp} \arrow[dl, "1\times\gamma^{\perp}"'] \arrow[dr, "1\times !"] & \\
1\times A \arrow[rrruuu, dash, bend left, "="] & & A\times A^{\perp}\times A \arrow[rr, dash, "a"] & & A\times(A^{\perp}\times A) & & A\times 1
\end{tikzcd}$$

Note that the above diagram is just a witness in $\mathbf{X}$ to the process equation $1\otimes\tau_A;\delta;\tau_A^{\perp}\otimes 1 \;=\; c : I\otimes A \to A\otimes I$.

Proposition 2.4.1 *If $\mathbf{X}$ has finite products then $\mathrm{Proc}(\mathbf{X}, \mathcal{X})$ is compact-closed for any cover system $\mathcal{X}$.*

Proof. Take $\_^{\perp}$ on $\mathbf{X}$ to be the identity, FA to be A and γ to be the diagonal Δ. The cover morphism x is then given by the isomorphism $u^{-1} : A \to 1\times A$. □

For example, the categories $\mathrm{Span}(\mathbf{X})$ and $\mathrm{Rel}(\mathbf{E})$ of spans and relations are compact-closed for finitely complete $\mathbf{X}$ and regular $\mathbf{E}$. Furthermore, $\mathrm{SProc}(\mathbf{X})$ is compact-closed for any finitely complete $\mathbf{X}$.

2.4.2 Biproducts

Lindner [25] observed that the span category of any lextensive category has biproducts: since the coproduct functor is stable and the associated natural transformations are cartesian, the span category has finite coproducts which are simultaneously products by self duality. To obtain biproducts in a process category it is then sufficient that coproducts preserve the cover system.

Proposition 2.4.2 *For lextensive* $\mathbf{X}$ *with cover system* $\mathcal{X}$*, the process category* $\mathrm{Proc}(\mathbf{X}, \mathcal{X})$ *has finite biproducts provided* $+$ *preserves* $\mathcal{X}$*.*

As discussed in section 2.1 the category $\mathrm{Tran}(\mathbf{X})$ of transition systems does not have extensive coproducts due to the fact that the initial state is (nontrivially) reachable. However, we can obtain an extensive coproduct in a related category of transition systems which have their initial state separate from the rest of the state space:

Example 2.4.3 $\mathrm{STran}(\mathbf{X})$ *is the category of models in* $\mathbf{X}$ *of the following sketch*

$$\begin{tikzcd} & P^0 \arrow[dl, "m^0"'] \arrow[dr, "\alpha^0"] & \\ \Sigma & & S \end{tikzcd} \qquad \begin{tikzcd} & P \arrow[dl, "m"'] \arrow[dr, "\alpha"] & \\ S\times\Sigma & & S \end{tikzcd}$$

The cover system $\exists^{\mathcal{X}}$ in this model category is the class of morphisms which are cartesian for both m^0 and $m; p_0$. Since the extensive coproducts (given componentwise) in $\mathrm{STran}(\mathbf{X})$ will preserve $\exists^{\mathcal{X}}$ whenever coproducts in $\mathbf{X}$ preserve $\mathcal{X}$,

Example 2.4.4 $\mathrm{Proc}(\mathrm{STran}(\mathbf{X}), \exists^{\mathcal{X}})$ *has biproducts whenever coproducts in* $\mathbf{X}$ *preserve* $\mathcal{X}$*.*

Proof. To see that coproducts in $\mathrm{STran}(\mathbf{X})$ preserve $\mathcal{X}$, it is enough to note that coproducts in $\mathbf{X}$ preserve $\mathcal{X}$-pullbacks since they preserve both pullbacks and $\mathcal{X}$-morphisms. □

Since coproducts in a regular category $\mathbf{E}$ preserve regular epimorphisms one always has biproducts in $\mathrm{SProc}(\mathbf{E})$.

2.4.3 Storage

An inexpensive way of obtaining the exponential type is to use the Fock construction [5]: in a compact-closed category with finite / countable biproducts $!A \stackrel{def}{=} \Sigma_{n\in\omega} \otimes^n_s A$, where $\otimes^n_s A$ is the n-th symmetric tensor power of A obtained by coequalizing the group $\mathcal{S}_n$ of permutations on $\otimes^n A$

$$\begin{tikzcd} \otimes^n A \arrow[r, shift left, "\sigma_1"] \arrow[r, shift right, "\sigma_{n!}"'] & \otimes^n A \arrow[r, two heads, "q"] & \otimes^n_s A \end{tikzcd}$$

In Rel, for example, these permutations and their coequalizers are given directly (i.e. as trivial spans) by the corresponding permutations and coequalizers on the product in **Set**:

$$\begin{tikzcd} A^n \arrow[r, shift left, "\sigma_1"] \arrow[r, shift right, "\sigma_{n!}"'] & A^n \arrow[r, two heads, "q"] & M_n(A) \end{tikzcd}$$

Given any $r : \otimes^n A \to B$ in Rel which satisfies $\forall \sigma \in \mathcal{S}_n.\ \sigma; r = r$, the mediating morphism $\otimes_s^n A \to B$ is given by $q^\circ; r$. Thus $!(\_)$ in Rel is given by the underlying multiset monad $M(A) \equiv \Sigma_{n\in\omega} M_n(A)$ on **Set**: the functor $!(\_)$ takes a relation (f_0, f_1) to the relation (Mf_0, Mf_1).

Unfortunately the Fock construction fails in SProc, contrary to suggestions in [2]. To see why, let A and B be the trace specifications generated by the regular expressions $(a_1|a_2)^*$ and b^* respectively. Forming the coequalizer $q : A{\times}A \to M_2(A)$ of the identity and symmetry morphisms in the category of traces gives the obvious candidate for the coequalizer in SProc: the process $q \stackrel{def}{=} 1_{A\otimes A}[((a_1, a_2), (a_1, a_2)) \mapsto ((a_1, a_2), \{\!|a_1, a_2|\!\})]$, using the notation of [2]. The process $r : A \otimes A \to B$ given by the transition system

$$\begin{tikzcd} & \bullet \arrow[r, "{(a_1,a_2),b}"] & \bullet \\ i \arrow[ur, "{(a_1,a_2),b}"] \arrow[dr, "{(a_2,a_1),b}"'] & & \\ & \bullet \arrow[r, "{(a_2,a_1),b}"'] & \bullet \end{tikzcd}$$

satisfies $c; r = r$. However, the composite process $q; q^\circ; r$ is not bisimilar (or even trace equivalent) to r since the trace $((a_1, a_2), b)((a_2, a_1), b)$ is admitted only by the former.

The fact that the Fock construction fails does not mean one cannot use multisets to interpret the exponential type. Any monad in SProc is simultaneously a comonad and so provides a potential candidate to model $!(\_)$. Thus we now consider how to obtain a suitable monad in a model category for SProc.

Note in **Set** the natural transformation q from the list monad L to the multiset monad which is given by the sum $\Sigma_{n\in\omega} q_n$ of coequalizers of $\mathcal{S}_n$

$$\begin{tikzcd} A \arrow[r, "inj"] \arrow[d, "="'] & LA \arrow[d, "q"] & LLA \arrow[l, "flatten"'] \arrow[d, "Lq;q"] \\ A \arrow[r, "inj"'] & MA & MMA \arrow[l, "flatten"] \end{tikzcd} \qquad \begin{tikzcd} 1 \arrow[r, "nil"] \arrow[d, "="'] & LA \arrow[d, "q"] & LA\times LA \arrow[l, "append"'] \arrow[d, "q\times q"] \\ 1 \arrow[r, "empty"'] & MA & MA\times MA \arrow[l, "union"] \end{tikzcd}$$

This natural transformation is pointwise a regular epimorphism and is also $\mathcal{E}$-cartesian. By lemma 2.3.9, the fact that L is a cartesian monad implies that M is an $\mathcal{E}$-cartesian monad. Furthermore, the isomorphisms $MA{\times}MB \to M(A{+}B)$ and $1 \to M0$ in **Set** give isomorphisms $!A{\otimes}!B \to !(A{\times}B)$ and $I \to !1$ in the category of relations.

Unfortunately $!(\_)$ is not a functor in SProc. To see that $!(\_)$ does not preserve composition, consider the specifications $A \equiv (a_1|a_2)^*$ and $B \equiv (b_1|b_2)^*$ and the process $f : A \to B$ given by the transition system

$$\begin{tikzcd} & \bullet \arrow[r, "{a_1,b_1}"] & \bullet \\ i \arrow[ur, "{a_1,b_1}"] \arrow[dr, "{a_2,b_2}"'] & & \\ & \bullet \arrow[r, "{a_2,b_2}"'] & \bullet \end{tikzcd}$$

The processes $!f;!f^\circ$ and $!(f;f^\circ)$ are not bisimilar as only the former can deadlock after exhibiting the action $(\{\!|a_1, a_2|\!\}, \{\!|a_1, a_2|\!\})$.

The abovementioned failing corresponds to the fact that the multiset functor is not an $\exists^{\mathcal{E}}$-stable functor on the model category for SProc. The above example, however, does not refute the possibility that $!(\_)$ is a functor under a more generous equivalence. Indeed, if we quotient SProc by trace equivalence rather than bisimulation the resulting process category does have a model of $!(\_)$ given by multisets. This fact is most easily seen by taking the model category of trees.

Note that Tree(**Set**) is a topos [20] and that lists and multisets are inherited pointwise from **Set**: any functor $F : \mathbf{X} \to \mathbf{Y}$ gives a functor $\mathrm{Tree}(F) : \mathrm{Tree}(\mathbf{X}) \to \mathrm{Tree}(\mathbf{Y})$

$$\begin{tikzcd} FA_0 \arrow[d, "Ff_0"'] & FA_1 \arrow[l, "Fa_0"'] \arrow[d, "Ff_1"'] & \cdots \arrow[l, "Fa_1"'] \\ FB_0 & FB_1 \arrow[l, "Fb_0"] & \cdots \arrow[l, "Fb_1"] \end{tikzcd}$$

and for any $\alpha : F \Rightarrow G : \mathbf{X} \to \mathbf{Y}$ the family $\{(\alpha_{A_0}, \alpha_{A_1}, \ldots) \mid A \in \mathrm{Tree}(\mathbf{Y})\}$ is a natural transformation $\mathrm{Tree}(F) \Rightarrow \mathrm{Tree}(G)$. In Tree(**Set**), the inherited $\mathrm{Tree}(q) : \mathrm{Tree}(L) \to \mathrm{Tree}(M)$ is in $\mathcal{E}$ and is $\mathcal{E}$-cartesian as this is true pointwise in **Set**:

$$\begin{tikzcd} & & \cdot \arrow[dl] & & & \cdot \arrow[dl] \\ & LA_1 \arrow[rrr] \arrow[dl, "La_0"'] \arrow[ddd] & & & LB_1 \arrow[dl, "Lb_0"] \arrow[ddd] & \\ LA_0 \arrow[rrr, "Lf_0"] \arrow[ddd, "q_{A_0}"'] & & & LB_0 \arrow[ddd, "q_{B_0}"] & & \\ & & \cdot \arrow[dl] & & & \cdot \arrow[dl] \\ & MA_1 \arrow[rrr] \arrow[dl, "Ma_0"'] & & & MB_1 \arrow[dl, "Mb_0"] & \\ MA_0 \arrow[rrr, "Mf_0"'] & & & MB_0 & & \end{tikzcd}$$

This makes the monad Tree(M) and associated monoid structure $\mathcal{E}$-cartesian. Again the isomorphisms $1 \to M0$ and $MA{\times}MB \to M(A{+}B)$ given pointwise induce the expected isomorphisms in the process category.

Proposition 2.4.5 SProc_T *is a linear category.*

This construction can be performed in a more general setting. For example, the multiset functor exists in any locos in which each object can be totally ordered.

2.5 Examples

In this section we discuss some process categories which result by varying the parameters to the process construction. First, by choosing appropriate cover systems for process legs, we construct processes upon transition systems which are deterministic and (in a certain sense) live. We then consider the two models of games.

2.5.1 Functional processes

Consider a span in $\mathrm{Tran}(\mathbf{X})$ whose left leg belongs to the cover system $\exists^{\mathcal{M}}$ of local monics.

$$\begin{tikzcd} & P \arrow[dl, "f"'] \arrow[dr, "g"] & \\ A & & B \end{tikzcd}$$

Since f is in $\exists^{\mathcal{M}}$, every transition from a state $f(s)$ in A has at most one corresponding transition from state s in P. One may view the interaction at interface A as input which determines the state change of the process and thus the output at interface B. With this in mind, we define the category of *deterministic* synchronous processes as follows:

$$\mathrm{DSProc}(\mathbf{X}) \stackrel{def}{=} \mathrm{Proc}^{\exists^{\mathcal{M}}}(\mathrm{Tran}(\mathbf{X}), \exists^{\mathcal{R}})$$

Note that the cover system for equivalence is actually forced to be the local isomorphisms: since $\exists^{\mathcal{M}}$ is left-factor closed all 2-cells in $\mathrm{Span}^{\exists^{\mathcal{M}}}(\mathrm{Tran}(\mathbf{X}), \exists^{\mathcal{R}})$ must belong to the intersection of $\exists^{\mathcal{R}}$ and $\exists^{\mathcal{M}}$, which is $\exists^{\mathcal{I}}$.

One might wish for a more relaxed definition of determinism in which state change is determined by presenting actions at both interfaces simultaneously. It is easily seen, however, that this property is not preserved by process composition [27].

Consider now the effect of requiring that spans take their left leg from the the cover system $\exists^{\mathcal{I}}$ of local isomorphisms: every transition from a state $f(s)$ in A has exactly on corresponding transition from state s in P. Such a process is not only deterministic, but implements it's interface completely: it can halt only at those states where the interface specifies no available actions. Thus we define the category of *functional* synchronous processes:

$$\mathrm{FSProc}(\mathbf{X}) \stackrel{def}{=} \mathrm{Proc}^{\exists^{\mathcal{I}}}(\mathrm{Tran}(\mathbf{X}), \exists^{\mathcal{I}})$$

We will see later that this process category is essentially the category of trees in $\mathbf{X}$, and that the former category is essentially the category of partial maps on trees.

2.5.2 Chu spaces

The category of Chu spaces (or $\mathrm{Chu}(\mathbf{Set}, 2)$) advocated by Pratt [30] has as objects A relations $R_A \subseteq E_A \times S_A$, and as arrows $A \to B$ pairs $(f : E_A \to E_B, g : S_B \to S_A)$ of functions satisfying the condition $(e, g(s)) \in R_A$ if and only if $(f(e), s) \in R_B$. Composition and identities are given componentwise in $\mathbf{Set}$.

The stated condition on morphisms is equivalently expressed by requiring the following pullback:

$$\begin{tikzcd}
R_A \arrow[d, "r_A"'] & \cdot \arrow[l] \arrow[r] \arrow[d] \arrow[dl, phantom, "\llcorner", very near start] \arrow[dr, phantom, "\lrcorner", very near start] & R_B \arrow[d, "r_B"] \\
E_A \times S_A & E_A \times S_B \arrow[l, "1\times g"] \arrow[r, "f\times 1"'] & E_B \times S_B
\end{tikzcd}$$

Consequently, the category of Chu spaces may be presented as a process category upon the category of models given by the sketch:

$$\begin{tikzcd}
R \arrow[d, "r"] \\
E \times S
\end{tikzcd}$$

The cover system for left legs is given by the morphisms which are cartesian for r and have their E-component an isomorphism, while the cover system for right legs are the morphisms which are cartesian for r and have their S-component an isomorphism. The cover system for equivalence is then forced to be the isomorphisms.

2.5.3 Simultaneous games

Consider the form of 2-player game in which both players move simultaneously (reference ???). Such games can be modelled as transition systems which are labelled by pairs taken from two independent alphabets.

Example 2.5.1 *For* $\mathbf{X}$ *a category with finite limits, define* $\mathrm{SGame}(\mathbf{X})$ *to be the category of models in* $\mathbf{X}$ *of the following sketch:*

$$
\begin{tikzcd}
& & P \arrow[dl, "p^\Sigma"'] \arrow[dr, "p^\Lambda"] \arrow[rr, bend left, "t"] & & S & 1 \arrow[l, "i"'] \\
& P^\Sigma \arrow[dl, "n^\Sigma"'] \arrow[dr, "s^\Sigma"] & & P^\Lambda \arrow[dl, "s^\Lambda"'] \arrow[dr, "n^\Lambda"] & & \\
\Sigma & & S & & \Lambda &
\end{tikzcd}
$$

Here the morphisms $\langle s^\Sigma, n^\Sigma \rangle$ and $\langle s^\Lambda, n^\Lambda \rangle$ indicate the permissible moves for player and opponent respectively, and t determines how each pair of independent moves (viz. P) effects the state of the game.

Consider as cover systems for span construction the classes of morphisms $\mathcal{L}_0$ and $\mathcal{L}_1$ which are cartesian for the arrows s^Σ and s^Λ respectively. A process

$$
\begin{tikzcd}
& P \arrow[dl, "f"'] \arrow[dr, "g"] & \\
A & & B
\end{tikzcd}
$$

constructed in this manner has the property that at any state s in P, each Σ-move permissible at state $f s$ in A has a unique corresponding move permitted at s and each Λ-move permissible at state gs in B has a unique corresponding move permitted at s. Together with the condition of independence in P, such processes are deterministic and deadlock-free: the presentation of inputs at each interface (viz. Σ_A and Λ_B) determines a unique evolution of the process which produces outputs at each interface (viz. Λ_A and Σ_B). The cover system for process equivalence is forced to be the local isomorphisms $\exists^{\mathcal{I}}$, which is the intersection of $\mathcal{L}_0$ and $\mathcal{L}_1$. Note that in this process category, the Σ-strategies for any game A are exactly the morphisms $A \to I$ (where I is game which is final in the model category SGame), while the Λ-strategies are just the processes $I \to A$.

By weaking the cover systems for span construction to be only $\mathcal{R}$-cartesian for $m^\Sigma; p_0$ and $m^\Lambda; p_0$ respectively, one maintains freedom from deadlock while allowing processes to be nondeterministic. In this case one can use the cover system $\exists^{\mathcal{R}}$ to obtain bisimulation as process equivalence.

Concerning the structure of this process category, note that there is a covariant involution $\_^\perp$ on the model category which switches the roles of the two players: i.e. $P^\Sigma \mapsto P^\Lambda$ and $P^\Lambda \mapsto P^\Sigma$. The product in the model category provides a symmetric tensor $\_ \otimes \_$ on the process category which is equal to the induced cotensor $A \oplus B \stackrel{def}{=} (A^\perp \otimes B^\perp)^\perp$. One then wonders whether or not the process category is compact-closed using the technique of section 2.4.1. The complementation process $I \to A \otimes A^{\perp}$ must translate an input pair (λ, σ) to the output pair (σ, λ): an obvious candidate for the apex of this process

$$\begin{tikzcd} & FA \arrow[dl, "!"'] \arrow[dr, "\gamma"] & \\ 1 & & A \times A^{\perp} \end{tikzcd}$$

is a relabelling of A in which the Σ-moves become $\Sigma_A \otimes \Lambda_A$ and the Λ-moves become $\Lambda_A \otimes \Sigma_A$, with both permission sets given by the permission set of the transition system A. The existence of a morphism $FA \to 1$ requires that A be *progressive* in that there are transitions from every state. The condition of independence on FA, however, requires that it provide transitions for all pairings of permissible moves, thus precluding the commuting requirements of the complementation process.

Another disappointing note is that these process categories do not have biproducts since the coproduct injections do not belong to the specified cover systems.

2.5.4 Interleaved games

Consider the form of 2-player game in which either player may initiate play, but afterwards the moves of each player alternate. Such games have become popular recently in the area of programming language semantics [4].

For $\mathbf{X}$ a category with finite limits:

Example 2.5.2 Game($\mathbf{X}$) *is the category of models in* $\mathbf{X}$ *of the following sketch:*

$$\begin{tikzcd} & & P^0 \arrow[dl, "s^0"'] \arrow[d, "\lambda^0"] \arrow[drr, "t^0"] & & & \\ 1 \arrow[r, "i^0"] & S^0 & \Sigma^0 & \Sigma^1 & S^1 & 1 \arrow[l, "i^1"'] \\ & & & P^1 \arrow[ull, "t^1"] \arrow[u, "\lambda^1"] \arrow[ur, "s^1"'] & & \end{tikzcd}$$

Thus we view a game as a transition system which has been "pulled apart": the 0-tagged components corresponding to the *player* side and the 1-tagged components corresponding to the *opponent* side.

There is an involution $\_^{\perp}$ on the model category which switches the roles of player and opponent, and taking $\mathcal{L}$ be the class of morphisms which are cartesian for s^0 we obtain the process category

$$\mathrm{Strat}(\mathbf{X}) \stackrel{def}{=} \mathrm{Proc}^{\mathcal{L},\mathcal{L}^{\perp}}(\mathrm{Game}(\mathbf{X}))$$

of games and synchronous strategies. Like the process category of section 2.5.3, the morphisms of $\mathrm{Strat}(\mathbf{X})$ can be seen as deterministic and deadlock-free processes.

The process category $\mathrm{Strat}(\mathbf{X})$ admits the standard game theoretic tensor. The game $A \otimes B$ allows games A and B to be played concurrently by interleaving: the player who moves first can do so in either component; afterwards the opponent may move in either component, while the player must move in the component where the opponent moved previously. The functor $\_\otimes\_$ is given by the following operation on the model category $\mathrm{Game}(\mathbf{X})$,

$$\begin{tikzcd}
 & & K_A\times 1+1\times K_B+(P^0_A\times S^1_B+S^1_A\times P^0_B) \arrow[dl, "!+(s^0_A\times 1+1\times s^0_B)"'] \arrow[d] \arrow[drr, "\langle (j_A;t^0_A)\times i^1_B \mid i^1_A\times(j_B;t^0_B) \mid \langle t^0_A\times 1 \mid 1\times t^0_B\rangle\rangle"] & & & \\
1 \arrow[r, "b_0"'] & 1+(S^0_A\times S^1_B+S^1_A\times S^0_B) & \Sigma^0_A+\Sigma^0_B & \Sigma^1_A+\Sigma^1_B & S^1_A\times S^1_B & 1 \arrow[l, "\langle i^1_A, i^1_B\rangle"] \\
 & & & P^1_A\times S^1_B+S^1_A\times P^1_B \arrow[ull, "\langle t^1_A\times 1 \mid 1\times t^1_B\rangle;b_1"] \arrow[u] \arrow[ur, "\langle s^1_A\times 1 \mid 1\times s^1_B\rangle"'] & &
\end{tikzcd}$$

where the labelling components are $p_0{+}p_1 + (p_0{+}p_1); \langle k_A{+}k_B \mid \lambda_A{+}\lambda_B\rangle$ and $p_0 + p_0; \lambda^1_A + \lambda^1_B$, and the initial player transitions of each game are extracted as follows:

$$\begin{tikzcd}
K \arrow[rr, "j"] \arrow[d, "k"'] \arrow[drr, phantom, "\lrcorner", very near start] & & P \arrow[d, "\langle s,\lambda\rangle"] \\
\Sigma \arrow[r, "u"'] & 1\times\Sigma \arrow[r, "i\times 1"'] & S\times\Sigma
\end{tikzcd}$$

The unit for the tensor is given by the final object in $\mathrm{Game}(\mathbf{X})$. Note that although $\_\otimes\_$ is an associative bifunctor on the model category $\mathrm{Game}(\mathbf{X})$, it is not a tensor since the unit elimination morphism is not an isomorphism — it is, however, a cover morphism strong enough to make it an isomorphism in the process category (see secion 4.2).

The tensor $\_\otimes\_$ and involution $\_^{\perp}$ give a cotensor $\oplus$ in the standard way, and these two tensors are linked by a linear distribution.

Claim 2.5.3 *For* $\mathbf{X}$ *a lextensive category,* $\mathrm{Strat}(\mathbf{X})$ *is linearly distributive.*

Although it has a similar flavor, Strat(**Set**) is not the category of games and strategies of Abramsky and Jagadeesan [3]. In particular, implementing the complementation process $I \to A \oplus A^{\perp}$ necessary for $*$-autonomy requires asynchrony.

2.6 Summary

We have presented the construction of a process category as a category of generalized relations: given a category of models of processes and a cover system for behavioral equivalence, a category of processes is formed as a span category quotiented by the cover system. Choosing additional cover systems to restrict the model morphisms used to construct spans provides a means of imposing on processes conditions which are preserved by composition: specifically, we have seen examples of the construction of deterministic and/or deadlock-free processes.

We have observed that much of the structure of a process category can be seen to arise from related structure in a model category, and that the construction is indeed a 2-functor. Thus the task of establishing functorial properties of upon process categories is largely the task of demonstrating that the underlying structure lies in the domain of the 2-functor. We have seen how this applies to establishing the presence of linear type structure in a process category, but it is also useful for inferring relationships between process categories: in chapter 4 we show how adjunctions between various categories of transition systems induce equivalence of the associated process categories.

Chapter 3

Adding asynchrony to model categories

This chapter presents an approach to obtaining asynchronous process categories: one uses a monad to add delay to a synchronous model category, and then performs the process construction upon the Kleisli category of that delay monad. This approach stems naturally from the desire to calculate at the level of the model category, and raises several technical questions: When does the Kleisli category have enough limits to support the process construction? How does one obtain a cover system which "ignores" delay? When is the cover-stable structure of the synchronous model category preserved by the Kleisli construction?

We begin with a review of the Kleisli construction and the conditions under which the Kleisli category inherits structure from it's underlying category. We then isolate a class of monads whose Kleisli categories have pullbacks, and develop conditions under which these Kleisli categories inherit the cover-stable structure of the underlying category. The lifting of functors also provides a useful technique for obtaining cover systems on Kleisli categories.

To illustrate these techniques, we show how the delay monad on transition systems admits the construction of a category of asynchronous processes modulo weak bisimulation. We conclude with a discussion of asynchrony in categories of games, where the notion of delay is more delicate.

3.1 The Kleisli construction

Recall that a monad on a category $\mathbf{X}$ consists of a functor $T : \mathbf{X} \to \mathbf{X}$ together with natural transformations $\eta : Id \Rightarrow T$ and $\mu : TT \Rightarrow T$ — the unit and multiplication — satisfying the following equations:

$$\begin{tikzcd}
& TA \arrow[dl, "T\eta"'] \arrow[d, "="] \arrow[dr, "\eta_T"] & \\
T^2A \arrow[r, "\mu"'] & TA & T^2A \arrow[l, "\mu"]
\end{tikzcd}
\qquad
\begin{tikzcd}
T^3A \arrow[r, "T\mu"] \arrow[d, "\mu"'] & T^2A \arrow[d, "\mu"] \\
T^2A \arrow[r, "\mu"'] & TA
\end{tikzcd}$$

A standard example is the monad of exceptions: if $\mathbf{X}$ is a category with finite coproducts and X is an object of $\mathbf{X}$, then $(\_ + X,\ b_0,\ a; 1{+}\nabla)$ is a monad.

Given a monad (T, η, μ) on $\mathbf{X}$, one forms the Kleisli category $\mathbf{X}_T$ with the same objects as $\mathbf{X}$, but with morphisms $A \to B$ given by the morphisms $A \to TB$ of $\mathbf{X}$. The identity id_A in $\mathbf{X}_T$ is given by η_A, while the composition of morphisms $f : A \to B$ and $g : B \to C$ in $\mathbf{X}_T$ is given by $f; Tg; \mu$ in $\mathbf{X}$.

There is an adjunction $(\eta, \epsilon) : F_T \dashv U_T : \mathbf{X} \to \mathbf{X}_T$ associated with the Kleisli construction: the free functor F_T takes $f : A \to B$ in $\mathbf{X}$ to $f; \eta$, the underlying functor U_T takes $g : A \to B$ in $\mathbf{X}_T$ to $Tg; \mu$, and the counit ϵ at A is given by id_{TA} in $\mathbf{X}$.

Suppose that (T, η, μ) and (S, ι, ν) are monads on categories $\mathbf{X}$ and $\mathbf{Y}$ respectively. If $G : \mathbf{X} \to \mathbf{Y}$ is a functor and $\lambda : GT \Rightarrow SG$ is a natural transformation, the operation G_λ which takes a morphism $f : A \to TB$ of $\mathbf{X}$ to the morphism $Gf; \lambda : GA \to SGB$ in $\mathbf{Y}$ is a functor $\mathbf{X}_T \to \mathbf{Y}_S$ if and only if

$$\begin{tikzcd}
& GA \arrow[dl, "G\eta"'] \arrow[dr, "\iota"] & \\
GTA \arrow[rr, "\lambda"'] & & SGA
\end{tikzcd}
\qquad
\begin{tikzcd}
GTTA \arrow[r, "\lambda"] \arrow[d, "G\mu"'] & SGTA \arrow[r, "S\lambda"] & SSGA \arrow[d, "\nu"] \\
GTA \arrow[rr, "\lambda"'] & & SGA
\end{tikzcd}$$

commutes for all A in $\mathbf{X}$. Such a λ is called a *distribution* for G over T and S.

For $\mathbf{X}$ a category with coproducts, the canonical isomorphism $\tau^+ : A{+}X{+}Y \to A{+}Y{+}X$ is a distribution for the functor $\_{+}Y$ over the monad $\_{+}X$. In a distributive category, a distribution for the functor $\_{\times}Y$ over $\_{+}X$ is given by the morphism $d; 1{\times}p_1 : (A{+}X){\times}Y \to A{\times}Y + X$. In both cases these tensor strengths induce distributions for the coproduct and product functors over the monad $\_+X$ as follows:

$$\begin{tikzcd}
TA\otimes TB \arrow[rrr, dashed] \arrow[d, "\tau"'] & & & T(A\otimes B) \\
T(A\otimes TB) \arrow[r, "Tc"'] & T(TB\otimes A) \arrow[r, "T\tau"'] & T^2(B\otimes A) \arrow[r, "T^2c"'] & T^2(A\otimes B) \arrow[u, "\mu"']
\end{tikzcd}$$

Note that distributions compose: if $\lambda : GT \Rightarrow SG$ and $\kappa : HS \Rightarrow RH$ are distributions, then $H\lambda;\kappa : HGT \Rightarrow RHG$ is a distribution. For example, one obtains a distribution $\tau{+}1;\tau$: $A{+}X{+}X{+}Y \to A{+}Y{+}Y{+}X$ for the functor $\_+Y+Y$ over the monad $\_+X$.

A special case of a distribution is when the "target" monad is the identity: for T a monad on $\mathbf{X}$ and $G : \mathbf{X} \to \mathbf{Y}$, a natural transformation $\alpha : GT \Rightarrow G$ for which

$$\begin{tikzcd}
 & GA \arrow[dl, "G\eta"'] \arrow[dr, "="] & \\
GTA \arrow[rr, "\alpha"'] & & GA
\end{tikzcd}
\qquad
\begin{tikzcd}
GTTA \arrow[r, "\alpha"] \arrow[d, "G\mu"'] & GTA \arrow[r, "\alpha"] & GA \arrow[d, "="] \\
GTA \arrow[rr, "\alpha"'] & & GA
\end{tikzcd}$$

is called a T-*action* for G.

If G and H are functors $\mathbf{X} \to \mathbf{Y}$ with distributions λ and κ respectively, a natural transformation $\alpha : G \Rightarrow H$ induces a natural transformation $\tilde{\alpha} : G_\lambda \Rightarrow H_\kappa$ given componentwise by $\alpha_A;\iota_{HA}$ in $\mathbf{Y}$ if and only if $\alpha;\kappa = \lambda; S\alpha$. In this case we will say that α *respects* the distributions λ and κ.

For example, the diagonal Δ in a distributive category lifts over the monad $\_+X$, but the projections do not. In addition,

Example 3.1.1 *In a category with coproducts, the injections and coidagonal lift over the monad* $\_+X$*, as do the unit and multiplication of* $\_+X$*:*

$$\begin{tikzcd}
A{+}X \arrow[r, "b_0"] \arrow[d, "="'] & A{+}X{+}Y \arrow[d, "\tau"] \\
A{+}X \arrow[r, "b_0+1"'] & A{+}Y{+}X
\end{tikzcd}
\quad
\begin{tikzcd}
A{+}X{+}(A{+}X) \arrow[r, "\nabla"] \arrow[d, "ex;1+\nabla"'] & A{+}X \arrow[d, "="] \\
A{+}A{+}X \arrow[r, "\nabla+1"'] & A{+}X
\end{tikzcd}
\quad
\begin{tikzcd}
A{+}X{+}Y{+}Y \arrow[r, "\mu"] \arrow[d, "\tau+1;\tau"'] & A{+}X{+}Y \arrow[d, "\tau"] \\
A{+}Y{+}Y{+}X \arrow[r, "\mu+1"'] & A{+}Y{+}X
\end{tikzcd}$$

The Kleisli construction on a 2-category $\underline{\mathbf{X}}$ can be seen as a 2-functor into $\underline{\mathbf{X}}$: the domain is the 2-category Dist($\underline{\mathbf{X}}$) whose 0-cells are monads T in $\underline{\mathbf{X}}$, 1-cells $T \to S$ are given by distributions $\lambda : GT \Rightarrow SG$ of $\underline{\mathbf{X}}$, and 2-cells $\lambda \Rightarrow \kappa$ are given by natural transformations $\alpha : G \Rightarrow H$ of $\underline{\mathbf{X}}$ which respect the distributions. There is a related 2-category of arrows Lift($\underline{\mathbf{X}}$) whose 0-cells are again monads in $\underline{\mathbf{X}}$, 1-cells $T \to S$ are "liftings", or pairs (G, G') such that $F_T; G' = G; F_S$, and 2-cells are "pillows", or pairs (α, α') such that $F_T;; \alpha' = \alpha;; F_S$.

Theorem 3.1.2 Dist($\underline{\mathbf{X}}$) *is isomorphic to* Lift($\underline{\mathbf{X}}$).

The proof appears in [**?**] and is based on the fact that the 1-cells of Lift($\underline{\mathbf{X}}$) correspond exactly to distributions.

3.2 Stable monads

Here we examine the conditions under which one can lift cover-stable functors and natural transformations over stable monads. The conditions are, of course, refinements on the conditions for lifting fuctors and natural transformations over arbitrary monads.

3.2.1 Finite limits in Kleisli categories

By focusing on monads which already behave well with respect to pullbacks, we provide necessary and sufficient conditions to secure pullbacks in the Kleisli category.

Let $\mathbf{X}$ be a category with pullbacks. A monad (T, η, μ) on $\mathbf{X}$ is called a *club* when T is stable and η and μ are cartesian [24]. We say that a club is a *stable monad* when it's Kleisli category has pullbacks.

Proposition 3.2.1 *A club (T, η, μ) is a stable monad if and only if there exists a stable functor $P : \mathbf{X} \to \mathbf{X}$ and cartesian natural transformations α and $\beta : P \Rightarrow TT$ which make*

$$\begin{tikzcd}
TPA \arrow[r, "T\alpha"] \arrow[d, "T\beta"'] \arrow[dr, phantom, very near start, "\lrcorner"] & T^3A \arrow[r, "\mu"] & T^2A \arrow[dd, "\mu"] \\
T^3A \arrow[d, "\mu"'] & & \\
T^2A \arrow[rr, "\mu"'] & & TA
\end{tikzcd}$$

a pullback in $\mathbf{X}$.

Proof. Note that ϵ_A, the counit of the Kleisli adjunction, is taken by U_T to μ_A in $\mathbf{X}$. So if $\mathbf{X}_T$ has pullbacks, the pullback of ϵ_A along itself is taken by U_T to the diagram above. It is not difficult to show that P is a stable functor and that α and β are cartesian natural transformations.

Conversely, if P, α and β are as stated, a pullback of f and g in $\mathbf{X}_T$ is given by a pullback of $U_T f$ and $U_T g$ in $\mathbf{X}$ which lies in the image of U_T: forming the double pullback (x, y) of (Tg, β, α, Tf) makes

$$\begin{tikzcd}
TZ \arrow[rr, "Tx"] \arrow[dd, "Ty"'] \arrow[dr, phantom, "\lrcorner", very near start] & & T^2A \arrow[r, "\mu"] \arrow[d, "T^2f"'] & TA \arrow[d, "Tf"] \\
& TPC \arrow[r, "T\alpha"] \arrow[d, "T\beta"'] & T^3C \arrow[r, "\mu"] & T^2C \arrow[dd, "\mu"] \\
T^2B \arrow[r, "T^2g"] \arrow[d, "\mu"'] & T^3C \arrow[d, "\mu"'] & & \\
TB \arrow[r, "Tg"'] & T^2C \arrow[rr, "\mu"'] & & TC
\end{tikzcd}$$

a pullback in $\mathbf{X}$ (as T is stable and μ is cartesian), and thus (x, y) is a pullback of f and g in $\mathbf{X}_T$. □

For the monad $\_ + X$, the stable monad structure is given by the functor T^3 and the natural transformations $T\mu$ and $\tau; T\mu$ respectively, where τ is the canonical isomorphism $A + X + Y \to A + Y + X$.

Example 3.2.2 *For a lextensive category* $\mathbf{X}$*, the monad* $\_+X$ *is a stable monad for any* X*.*

Proof. The required pullback is constructed in the following diagram, where (to save space) we write the coproduct of objects as juxtaposition:

$$
\begin{tikzcd}
AXXXX \arrow[rr, "a+1+1"] \arrow[ddd, "\tau+1"'] & & A(XX)XX \arrow[r, "1+\nabla+1+1"] \arrow[d, "a"] & AXXX \arrow[r, "a"] \arrow[dr, "a"] & AX(XX) \arrow[r, "1+1+\nabla"] \arrow[d, equal] & AXX \arrow[d, equal] \\
 & & A(XX)(XX) \arrow[rr, "1+\nabla+1"] \arrow[d, "a"] & & AX(XX) \arrow[r, "1+1+\nabla"] \arrow[d, "a"] & AXX \arrow[d, "a"] \\
 & & A(XX(XX)) \arrow[rr, "1+(\nabla+1)"] \arrow[d, "1+a^{-1}"] & & A(X(XX)) \arrow[r, "1+(1+\nabla)"] & A(XX) \arrow[ddd, equal] \\
AXXXX \arrow[dd, "a+1+1"'] & (1) & A(XXXX) \arrow[d, "1+(\tau+1)"] & & & \\
 & & A(XXXX) \arrow[d, "1+a"] & & (2) & \\
A(XX)XX \arrow[r, "a"] \arrow[d, "1+\nabla+1+1"'] & A(XX)(XX) \arrow[r, "a"] \arrow[dd, "1+\nabla+1"] & A(XX(XX)) \arrow[rrr, "1+\nabla"] \arrow[dd, "1+(\nabla+1)"] & & & A(XX) \arrow[dddd, "1+\nabla"] \\
AXXX \arrow[dr, "a"] \arrow[dd, "a"'] & & & & & \\
 & AX(XX) \arrow[r, "a"] \arrow[dd, "1+1+\nabla"] & A(X(XX)) \arrow[dd, "1+(1+\nabla)"] & & & \\
AX(XX) \arrow[ur, equal] \arrow[d, "1+1+\nabla"'] & & & & & \\
AXX \arrow[r, equal] & AXX \arrow[r, "a"] & A(XX) \arrow[rrr, "1+\nabla"] & & & AX
\end{tikzcd}
$$

(1) and (2) are easily shown to commute and are thus pullbacks as opposing sides are isomorphisms.
□

Given pullbacks, one secures all finite limits with the addition of a final object. It is easily seen that

Proposition 3.2.3 *Z is final in $\mathbf{X}_T$ if and only if TZ is final in $\mathbf{X}$.*

Thus for lextensive $\mathbf{X}$, the Kleisli category $\mathbf{X}_{\_+1}$ has finite limits since $0+1$ is isomorphic to 1.

3.2.2 Cover systems upon Kleisli categories

Let $\mathbf{X}$ be a category with cover system $\mathcal{X}$. A stable monad T on $\mathbf{X}$ is called a *cover-stable* monad on $(\mathbf{X}, \mathcal{X})$, or simply an $\mathcal{X}$-stable monad, provided T preserves $\mathcal{X}$.

Given an $\mathcal{X}$-stable monad T on $\mathbf{X}$, there is a canonical cover system $\mathcal{X}_T$ upon $\mathbf{X}_T$ given by closing $F_T(\mathcal{X})$ with the isomorphisms of $\mathbf{X}_T$. However, if the isomorphims of $\mathbf{X}_T$ are only the lifted isomorphisms of $\mathbf{X}$ then $F_T(\mathcal{X})$ is already a cover system,

Proposition 3.2.4 *If T is a cover-stable monad satisfying $\mathcal{I}_{\mathbf{X}_T} = F_T(\mathcal{I}_{\mathbf{X}})$, then $F_T(\mathcal{X})$ is a cover system on $\mathbf{X}_T$.*

Proof. $\mathcal{X}_T$ is closed to composition as F_T is a functor. To see that $\mathcal{X}_T$ is closed to pullback, suppose x is in $\mathcal{X}$ and f is a morphism of $\mathbf{X}_T$. The following pullback in $\mathbf{X}$ corresponds to a pullback $g; F_T x = F_T y; f$ in $\mathbf{X}_T$:

$$\begin{tikzcd} \cdot \arrow[r, "Tg"] \arrow[d, "Ty"'] \arrow[dr, phantom, "\lrcorner", very near start] & \cdot \arrow[r, "\mu"] \arrow[d, "T^2x"'] \arrow[dr, phantom, "\lrcorner", very near start] & \cdot \arrow[d, "Tx"] \\ \cdot \arrow[r, "Tf"'] & \cdot \arrow[r, "\mu"'] & \cdot \end{tikzcd}$$

y is in $\mathcal{X}$ as T preserves and reflects $\mathcal{X}$, and thus $F_T y$ is in $\mathcal{X}_T$. $\square$

Where we have used the following observation:

Lemma 3.2.5 *A functor T of a club on $\mathbf{X}$ reflects any cover system $\mathcal{X}$ on $\mathbf{X}$.*

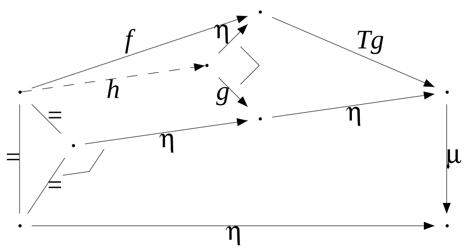
Proof. Since η is cartesian, the naturality square $f; \eta = \eta; Tf$ is a pullback and so f belongs to $\mathcal{X}$ whenever Tf belongs to $\mathcal{X}$. $\square$

Note that $F_T(\mathcal{X})$ does not generally inherit left-factor closure from $\mathcal{X}$. We will say that a cover-stable monad is *strongly* cover-stable whenever $\mathcal{I}_{\mathbf{X}_T} = F_T(\mathcal{I}_{\mathbf{X}})$.

Lemma 3.2.6 *An $\mathcal{X}$-stable monad T is strongly $\mathcal{X}$-stable provided η is monic and the following is a pullback for all A in $\mathbf{X}$:*

$$\begin{tikzcd} A \arrow[r, "\eta"] \arrow[d, equal] \arrow[drr, phantom, "\lrcorner", very near start] & TA \arrow[r, "\eta"] & T^2A \arrow[d, "\mu"] \\ A \arrow[rr, "\eta"'] & & TA \end{tikzcd}$$

Proof. Suppose f is an isomorphism with inverse g in $\mathbf{X}_T$. Then $f; Tg; \mu = \eta$ in $\mathbf{X}$ and as η is cartesian there is a map h such that $f = F_T(h)$:



Similarly, there exists k such that $g = F_T(k)$. Furthermore, $h; k = 1$ and $k; h = 1$ as $\eta = b_0$ is monic. $\square$

The exception monad (and thus the derived monads of delay on transition systems) are strongly cover-stable.

Example 3.2.7 *In a lextensive category* $\mathbf{C}$*, the monad* $\_+X$ *is strongly* $\mathcal{C}$*-stable provided coproducts in* $\mathbf{C}$ *preserve* $\mathcal{C}$*.*

Proof. Coproduct injections are monic in a lextensive category, and the pullback diagram of lemma 3.2.6 appears below:

$$\begin{tikzcd}
A \arrow[r, "b_0"] \arrow[d, equal] \arrow[dr, phantom, very near start, "\lrcorner"] & A+X \arrow[r, "b_0"] & A+X+X \arrow[d, "a"] \\
A \arrow[rr, "b_0"] \arrow[d, equal] \arrow[dr, phantom, very near start, "\lrcorner"] & & A+(X+X) \arrow[d, "1+\nabla"] \\
A \arrow[rr, "b_0"'] & & A+X
\end{tikzcd}$$

$\square$

Another way in which a Kleisli category inherits a cover system from its underlying category is through a stable monad action, which we will now investigate.

3.2.3 Lifting stable functors

Suppose now that T and S are cover-stable monads on $(\mathbf{X}, \mathcal{X})$ and $(\mathbf{Y}, \mathcal{Y})$ respectively, and that $G : (\mathbf{X}, \mathcal{X}) \to (\mathbf{Y}, \mathcal{Y})$ is a cover-stable functor.

Proposition 3.2.8 *If* $\lambda : GT \Rightarrow SG$ *is a* $\mathcal{Y}$*-cartesian natural transformation, then* $G_\lambda : (\mathbf{X}_T, \mathcal{X}_T) \to (\mathbf{Y}_T, \mathcal{Y}_T)$ *is a cover-stable functor if and only if* λ *is a distribution for* G *and*

$$\begin{tikzcd}
GP_TA \arrow[r, "G\mu_0"] \arrow[d, "G\mu_1"'] & GT^2A \arrow[r, "\lambda"] & SGTA \arrow[d, "S\lambda"] \\
GT^2A \arrow[d, "\lambda"'] & P_SGA \arrow[r, "\nu_0"] \arrow[d, "\nu_1"'] & S^2GA \\
SGTA \arrow[r, "S\lambda"'] & S^2GA &
\end{tikzcd}$$

is a $\mathcal{Y}$*-double pullback for all* A *in* $\mathbf{X}$*.*

Proof. ($\Rightarrow$) The pullback of ϵ_A along itself is taken by the composite $U_S G_\lambda$ to the following $\mathcal{Y}$-pullback:

$$\begin{tikzcd}
SGP_TA \arrow[r, "SG\mu_0"] \arrow[d, "SG\mu_1"'] & SGT^2A \arrow[r, "S\lambda"] & S^2GTA \arrow[r, "\nu"] \arrow[d, "S^2\lambda"'] \arrow[dr, phantom, "\lrcorner", very near start] & SGTA \arrow[d, "S\lambda"] \\
SGT^2A \arrow[d, "S\lambda"'] & SP_SGA \arrow[r, "S\nu_0"] \arrow[d, "S\nu_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & S^3GA \arrow[r, "\nu"] & S^2GA \arrow[dd, "\nu"] \\
S^2GTA \arrow[r, "S^2\lambda"] \arrow[d, "\nu"'] \arrow[dr, phantom, "\lrcorner", very near start] & S^3GA \arrow[d, "\nu"'] & & \\
SGTA \arrow[r, "S\lambda"'] & S^2GA \arrow[rr, "\nu"'] & & SGA
\end{tikzcd}$$

Since S reflects cover-pullbacks, the diagram in question is a $\mathcal{Y}$-double pullback.

($\Leftarrow$) If $h; f = k; g$ is an $\mathcal{X}_T$-pullback then as U_T preserves and T reflects cover pullbacks the following is an $\mathcal{X}$-double pullback:

$$\begin{tikzcd}
Z \arrow[rr, "h"] \arrow[dd, "k"'] & & TA \arrow[d, "Tf"] \\
& P_TC \arrow[r, "\mu_0"] \arrow[d, "\mu_1"'] & T^2C \\
TB \arrow[r, "Tg"'] & T^2C &
\end{tikzcd}$$

Let y be the induced $\mathcal{Y}$-map the the double pullback (p, q) inscribed in the following:

$$\begin{tikzcd}
GZ \arrow[rr, "Gh"] \arrow[dd, "Gk"'] & & GTA \arrow[r, "\lambda"] \arrow[d, "GTf"'] & SGA \arrow[d, "SGf"] \\
& GP_TC \arrow[r, "G\mu_0"] \arrow[d, "G\mu_1"'] & GT^2C \arrow[r, "\lambda"] & SGTC \arrow[d, "S\lambda"] \\
GTB \arrow[r, "GTg"] \arrow[d, "\lambda"'] & GT^2C \arrow[d, "\lambda"'] & P_SGC \arrow[r, "\nu_0"] \arrow[d, "\nu_1"'] & S^2GC \\
SGB \arrow[r, "SGg"'] & SGTC \arrow[r, "S\lambda"'] & S^2GC &
\end{tikzcd}$$

The Sy is the induced $\mathcal{Y}$-map to the pullback inscribe in the following:

$$\begin{tikzcd}
SGZ \arrow[rrr, "SGh"] \arrow[dr, "Sy"] \arrow[ddd, "SGk"'] & & & SGTA \arrow[r, "S\lambda"] & S^2GA \arrow[r, "\nu"] \arrow[d, "S^2Gf"'] \arrow[dr, phantom, "\lrcorner", very near start] & SGA \arrow[d, "SGf"] \\
& SP \arrow[urrr, "Sp"] \arrow[ddl, "Sq"] \arrow[dr, phantom, "\lrcorner", very near start] & & & & \\
& & & & S^2GTC \arrow[r, "\nu"] \arrow[d, "S^2\lambda"'] \arrow[dr, phantom, "\lrcorner", very near start] & SGTC \arrow[d, "S\lambda"] \\
SGTB \arrow[d, "S\lambda"'] & & & SP_SGC \arrow[r, "S\nu_0"] \arrow[d, "S\nu_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & S^3GC \arrow[r, "\nu"] & S^2GC \arrow[dd, "\nu"] \\
S^2GB \arrow[rr, "S^2Gg"] \arrow[d, "\nu"'] \arrow[drr, phantom, "\lrcorner", very near start] & & S^2GTC \arrow[r, "S^2\lambda"] \arrow[d, "\nu"'] \arrow[dr, phantom, "\lrcorner", very near start] & S^3GC \arrow[d, "\nu"] & & \\
SGB \arrow[rr, "SGg"'] & & SGTC \arrow[r, "S\lambda"'] & S^2GC \arrow[rr, "\nu"'] & & SGC
\end{tikzcd}$$

Note that $p; G_\lambda f = q; G_\lambda g$ is a pullback in $\mathbf{Y}_S$; as the following equations hold in $\mathbf{Y}$,

$$Gh; \lambda = y; p = y; \iota; Sp; \nu$$

$$Gk; \lambda = y; q = y; \iota; Sq; \nu$$

the image of $h; f = k; g$ under G_λ is a $\mathcal{Y}_S$-pullback. $\square$

We say that a natural transformation λ satisfying proposition 3.2.8 is a *cover-stable* distribution. The following shows that cover-stable distributions may be composed:

Lemma 3.2.9 *If $F_\lambda : (\mathbf{X}_T, \mathcal{X}_T) \to (\mathbf{Y}_S, \mathcal{Y}_S)$ and $G_\kappa : (\mathbf{Y}_S, \mathcal{Y}S) \to (\mathbf{Z}_R, \mathcal{Z}_R)$ are cover-stable functors, then $GF_{G\lambda;\kappa}$ is a cover-stable functor $(\mathbf{X}_T, \mathcal{X}_T) \to (\mathbf{Z}_R, \mathcal{Z}_R)$.*

Proof. The required diagram is as follows:

$$\begin{tikzcd}
\cdot \arrow[r, "GF\mu_0"] \arrow[d, "GF\mu_1"'] & \cdot \arrow[r, "G\lambda"] & \cdot \arrow[r, "\kappa"] \arrow[d, "GS\lambda"'] & \cdot \arrow[d, "RG\lambda"] \\
\cdot \arrow[d, "G\lambda"'] & \cdot \arrow[r, "G\nu_0"] \arrow[d, "G\nu_1"'] & \cdot \arrow[r, "\kappa"] & \cdot \arrow[d, "R\kappa"] \\
\cdot \arrow[r, "GS\lambda"] \arrow[d, "\kappa"'] & \cdot \arrow[d, "\kappa"'] & \cdot \arrow[r, "\sigma_0"'] \arrow[d, "\sigma_1"] & \cdot \\
\cdot \arrow[r, "RG\lambda"'] & \cdot \arrow[r, "R\kappa"] & \cdot &
\end{tikzcd}$$

$\square$

The following lemma is useful for deciding whether or not a distribution over an exception monad is cover stable.

Lemma 3.2.10 *If λ is a distribution for G over T and S such that*

$$\begin{tikzcd}
GT^2A \arrow[r, "\lambda"] \arrow[d, "G\mu"'] & SGTA \arrow[r, "S\lambda"] & S^2GA \arrow[d, "\nu"] \\
GTA \arrow[rr, "\lambda"'] & & SGA
\end{tikzcd}$$

is a pullback, then λ is cover-stable if and only if

$$\begin{tikzcd}
GT^3A \arrow[r, "G\tau^T"] \arrow[dd, "\lambda"'] & GT^3A \arrow[rr, "\lambda"] & & SGT^2A \arrow[d, "S\lambda"] \\
& & & S^2GTA \arrow[d, "S^2\lambda"] \\
SGT^2A \arrow[r, "S\lambda"'] & S^2GTA \arrow[r, "S^2\lambda"'] & S^3GA \arrow[r, "\tau^S"'] & S^3GA
\end{tikzcd}$$

is a $\mathcal{Y}$-pullback.

Proof. The $\mathcal{Y}$-double pullback of proposition 3.2.8 must be constructed as follows:

$$\begin{tikzcd}
GT^3A \arrow[rr, "Gt"] \arrow[ddd, "GT\mu"'] \arrow[ddddr, "\lambda"] & & GT^3A \arrow[r, "GT\mu"] \arrow[drr, "\lambda"] & GT^2A \arrow[rr, "\lambda"] & & SGTA \arrow[dddd, "S\lambda"] \\
& & & & SGT^2A \arrow[ur, "SG\mu"'] \arrow[d, "S\lambda"'] & \\
& & (1) & & S^2GTA \arrow[dd, "S^2\lambda"'] \arrow[r, phantom, "(2)"] & \\
GT^2A \arrow[dd, "\lambda"'] & & & & & \\
& SGT^2A \arrow[r, "S\lambda"] \arrow[dl, "SG\mu"] & S^2GTA \arrow[r, "S^2\lambda"] & S^3GA \arrow[r, "s"] \arrow[d, "S\nu"] & S^3GA \arrow[r, "S\nu"] & S^2GA \\
SGTA \arrow[rrr, "S\lambda"'] & & & S^2GA & &
\end{tikzcd}$$

□

For example, in the Kleisli category of the monad $\_+X$, the functor $\_+Y$ is stable:

Example 3.2.11 *For lextensive category* **C***, the tensorial strength τ^+ is a stable distribution for $\_+Y$ over $\_+X$. Consequently τ+1; τ is a stable distribution of $\_+Y+Y$ over $\_+X$.*

Proof. Diagram (1) is seen to commute by coherence for symmetric monoidal categories, and both (1) and (2) are pullbacks since τ is an isomorphism. $\square$

Unfortunately, most other examples of distributions over $\_+X$ are not cover stable: Both the product and coproduct functors have diagram (2) of lemma 3.2.10 a pullback, but (1) is neither a pullback nor an $\mathcal{R}$-pullback since the inscribed pullback is "larger".

Stable actions

As a source of functors from a Kleisli category to it's underlying category, actions are of interest in obtaining cover systems.

With T a stable monad on $\mathbf{X}$ and G a cover-stable functor $\mathbf{X} \to (\mathbf{Y}, \mathcal{Y})$, we note the following specialization of proposition 3.2.8:

Corollary 3.2.12 *If $\alpha : GT \Rightarrow G$ is a T-action for G, then G_α is functor $\mathbf{X}_T \to (\mathbf{Y}, \mathcal{Y})$ if and only if*

$$\begin{tikzcd}
GP_TA \arrow[r, "G\mu_0"] \arrow[d, "G\mu_1"'] & GT^2A \arrow[r, "\theta"] & GTA \arrow[dd, "\theta"] \\
GT^2A \arrow[d, "\theta"'] & & \\
GTA \arrow[rr, "\theta"'] & & GA
\end{tikzcd}$$

is a $\mathcal{Y}$-pullback for all A in $\mathbf{X}$:

An obvious example of a stable T-action for any stable monad T is the monad multiplication: μ lifts the functor T to the underlying functor U_T.

An example relevant to obtaining a cover system for weak bisimulation is given in **Set** by the list club ($\_^*$, inj, flatten) and the natrual transformation $\theta : (\_ + 1)^* \Rightarrow \_^*$

$$\begin{tikzcd}
(A+1)^* \arrow[r, "\langle inj|nil\rangle^*"] \arrow[dr, dashed] & A^{**} \arrow[d, "flatten"] \\
& A^*
\end{tikzcd}$$

which removes elements of X from strings.

Example 3.2.13 *In* **Set**, *$\theta : (\_ + 1)^* \Rightarrow (\_)^*$ is a stable action.*

Proof. The square required to be a pullback is as follows:

$$\begin{tikzcd}
(A+1+1+1)^* \arrow[r, "s^*"] \arrow[d, "(\mu+1)^*"'] & (A+1+1+1)^* \arrow[r, "(\mu+1)^*"] & (A+1+1)^* \arrow[r, "\theta"] & (A+1)^* \arrow[dd, "\theta"] \\
(A+1+1)^* \arrow[d, "\theta"'] & & & \\
(A+1)^* \arrow[rrr, "\theta"'] & & & A^*
\end{tikzcd}$$

It is easily seen to commute as each route simply strips the three distinct exceptions from each element of A+1+1+1. To see that it is a pullback, let h be the map to the pullback Q of θ_A and θ_A. Define $k : Q \to A$+1+1+1 as follows:

$$\begin{aligned}
k(\mathrm{nil}, \mathrm{nil}) &= \mathrm{nil} \\
k(* \operatorname{cons} \ell, \mathrm{nil}) &= *_1 \operatorname{cons} k(\ell, \mathrm{nil}) \\
k(* \operatorname{cons} \ell, a \operatorname{cons} m) &= *_1 \operatorname{cons} k(\ell, a \operatorname{cons} m) \\
k(\mathrm{nil}, * \operatorname{cons} m) &= *_2 \operatorname{cons} k(\mathrm{nil}, m) \\
k(a \operatorname{cons} \ell, * \operatorname{cons} m) &= *_2 \operatorname{cons} k(a \operatorname{cons} \ell, m) \\
k(* \operatorname{cons} \ell, * \operatorname{cons} m) &= *_3 \operatorname{cons} k(\ell, m) \\
k(a \operatorname{cons} \ell, a \operatorname{cons} m) &= a \operatorname{cons} k(\ell, m)
\end{aligned}$$

where $a \in \Sigma_A$. An induction on the structure of Q shows that k is the inverse of h. □

3.2.4 Lifting cartesian natural transformations

Suppose that G and H are cover-stable functors $(\mathbf{X}, \mathcal{X}) \to (\mathbf{Y}, \mathcal{Y})$ with cover-stable distributions λ and κ over cover-stable monads T and S respectively:

Proposition 3.2.14 *If $\alpha : G \Rightarrow H$ then $\tilde{\alpha} : G_\lambda \Rightarrow H_\kappa$ if and only if*

$$\begin{tikzcd}
GTA \arrow[r, "\alpha_{TA}"] \arrow[d, "\lambda_A"'] & HTA \arrow[d, "\kappa_A"] \\
SGA \arrow[r, "S\alpha_A"'] & SHA
\end{tikzcd}$$

is a $\mathcal{Y}$-pullback for all A.

Proof. ($\Rightarrow$) The naturality square of α associated with ϵ_A in $\mathbf{Y}_S$ is taken by U_S to the following $\mathcal{Y}$-pullback:

$$\begin{tikzcd}
SGTA \arrow[r, "S\lambda"] \arrow[d, "S\alpha"'] \arrow[dr, phantom, "(*)"] & S^2GA \arrow[r, "\nu"] \arrow[d, "S^2\alpha"'] \arrow[dr, phantom, "\lrcorner", very near start] & SGA \arrow[d, "S\alpha"] \\
SHTA \arrow[r, "S\kappa"'] & S^2HA \arrow[r, "\nu"'] & SHA
\end{tikzcd}$$

As ν is cartesian, (*) is a $\mathcal{Y}$-pullback and thus the required square is a $\mathcal{Y}$-pullback as S reflects $\mathcal{Y}$-pullbacks.

($\Leftarrow$) Let y be the $\mathcal{Y}$-map induced to the pullback (p, q) inscribed in:

$$\begin{tikzcd}
GA \arrow[r, "Gh"] \arrow[d, "\alpha"'] & GTB \arrow[r, "\lambda"] \arrow[d, "\alpha"'] & SGB \arrow[d, "S\alpha"] \\
HA \arrow[r, "Hh"'] & HTB \arrow[r, "\kappa"'] & SHB
\end{tikzcd}$$

Then Sy serves as the map induced to the pullback inscribed in the following,

$$\begin{tikzcd}
SGA \arrow[r, "SGh"] \arrow[dd, "S\alpha"'] \arrow[dr, "Sy"] & SGTB \arrow[r, "S\lambda"] & S^2GB \arrow[r, "\nu"] \arrow[dd, "S^2\alpha"'] \arrow[dr, phantom, "\lrcorner", very near start] & SGB \arrow[dd, "S\alpha"] \\
& \cdot \arrow[ur, "Sp"'] \arrow[dl, "Sq"] & & \\
SHA \arrow[r, "SHh"'] & SHTB \arrow[r, "S\kappa"'] & S^2HB \arrow[r, "\nu"'] & SHB
\end{tikzcd}$$

which implies $\tilde{\alpha}$ is $\mathcal{Y}_S$-cartesian. $\square$

For example, the unit and multiplication of the monad $\_+Y$ in the Kleisli category of $\_+X$ are cartesian natural transformations — the two relevant diagrams of example 3.1.1 are pullbacks since the distributions are isomorphisms.

3.2.5 2-categorical aspects

Technically, the Kleisli construction does not exist in the 2-category <u>**Cov**</u>: although the functors F_T and U_T are stable and the unit η is cartesian, the counit ϵ is not cartesian. For ϵ to be cartesian would require the image under U_T of a naturality square $\epsilon_A; f = FUf; \epsilon_B$

$$\begin{tikzcd}
T^2A \arrow[r, "T^2f"] \arrow[d, "\mu"'] & T^3B \arrow[r, "T\mu"] & T^2B \arrow[d, "\mu"] \\
TA \arrow[r, "Tf"'] & T^2B \arrow[r, "\mu"'] & TB
\end{tikzcd}$$

to be a pullback, which fails by proposition 3.2.1. However, the construction of Kleisli categories with cover systems can be characterized by an analogue of theorem 3.1.2.

Let CSDist($\underline{\mathbf{X}}$) be the subcategory of Dist($\underline{\mathbf{X}}$) whose 0-cells are cover-stable monads, 1-cells are cover-stable distributions and 2-cells respect the distributions in the sense of proposition 3.2.14.

Proposition 3.2.15 CSDist($\underline{\mathbf{X}}$) *is a 2-category.*

Proof. It is sufficient to note that composition of the 1-cells is well-defined by lemma 3.2.9. □

Let CSLift($\underline{\mathbf{X}}$) be the subcategory of Lift($\underline{\mathbf{X}}$) whose 0-cells are those monads which are cover-stable, 1-cells are liftings whose components are cover-stable, and 2-cells are pillows whose components are cover-cartesian. We can now state the analogue of theorem 3.1.2 which characterizes the construction of asynchronous model categories presented in this section.

Theorem 3.2.16 CSDist($\underline{\mathbf{X}}$) *is isomorphic to* CSLift($\underline{\mathbf{X}}$).

Proof. Any cover-stable lifting corresponds to a distribution, and thus the result is immediate from propositions 3.2.8 and 3.2.14. □

3.3 Example: Asynchronous processes

Here we apply the preceeding techniques to construct a category of asynchronous processes modulo weak bisimulation [27]. The construction provides an explicit connection between Abramsky's categories SProc and ASProc, and provides a functorial reconstruction of Milner's technique of for obtaining weak bisimulation from strong bisimulation.

3.3.1 The monad of delay

We begin with the delay monad on transition systems, which adds to a new label and an "idle" transition on that label at every state: This monad exists in the category of transition systems taken upon any lextensive category **X**.

Definition 3.3.1 *The monad D on* Tran(**X**) *has the following effect on a transition system A:*

$$\begin{tikzcd}[column sep=small] & R \arrow[dl, "r"'] \arrow[dr, "\alpha"] & & & & R+S \arrow[dl, "r+1"'] \arrow[ddrr, "\langle\alpha|1\rangle"] & & \\ S\times\Sigma & & S & \arrow[r, mapsto, "D"] & S\times\Sigma+S \arrow[d, "\sim"'] & & & \\ & & & & S\times(\Sigma+1) & & & S \end{tikzcd}$$

The monad structure is given componentwise by the identity monad and the exception monad: the unit η^D injects a transition system A into the non-idling portion of DA, and the multiplication μ^D collapses the two distinct idle actions of DDA into the single idle action of DA.

Proposition 3.3.2 *(D, η^D, μ^D) is an $\exists^{\mathcal{X}}$-stable monad on* $\mathrm{Tran}(\mathbf{X})$ *whenever coproducts in* $\mathbf{X}$ *preserve $\mathcal{X}$.*

Proof. It is easily shown that D is a functor and that η and μ are morphisms. The obligations of a stable monad are then given by the construction componentwise as the identity and exception monads.

To see that D is $\exists^{\mathcal{X}}$-stable: if $f : A \to B$ is in $\exists^{\mathcal{X}}$ then the following is a $\mathcal{X}$-pullback as $+$ is $\mathcal{X}$-stable and ∇ is cartesian:

$$\begin{tikzcd}
 & P_A+S_A \arrow[rrr, "f_P+f_S"] \arrow[d, "p_A+1"'] & & & P_B+S_B \arrow[d, "p_B+1"] \\
 & S_A\times\Sigma_A+S_A \arrow[dl, "\pi+1"'] \arrow[d, "\sim"] & & & S_B\times\Sigma_B+S_B \arrow[dl, "\pi+1"'] \arrow[d, "\sim"] \\
S_A+S_A \arrow[rrr, "f_S+f_S"] \arrow[dr, "\nabla"'] & S_A\times(\Sigma_A+1) \arrow[d, "\pi"] & & S_B+S_B \arrow[dr, "\nabla"'] & S_B\times(\Sigma_B+1) \arrow[d, "\pi"] \\
 & S_A \arrow[rrr, "f_S"'] & & & S_B
\end{tikzcd}$$

□

Consider a span in $\mathrm{Tran}(\mathbf{X})_D$ with interfaces A and B as seen in $\mathrm{Tran}(\mathbf{X})$:

$$\begin{tikzcd}
 & P \arrow[dl, "f"'] \arrow[dr, "g"] & \\
TA & & TB
\end{tikzcd}$$

As a process, each transition of the apex P can map to an idle transition at either or both interfaces. The transitions which map to idling are regarded as internal as they are neither seen in the interface nor have an effect on the interface.

Suppose that (P, f, g) and (Q, h, k) in $\mathrm{Tran}(\mathbf{Set})_D$ are spans which may be composed. Proposition 3.2.1 says that the apex of the composite span is the following transition system:

$$\begin{aligned}
&\{(p,q) \xrightarrow{a,b} (p',q') \mid p \xrightarrow{a} p' \in P \wedge q \xrightarrow{b} q' \in Q \wedge g(a) = h(b)\} \\
&\cup \{(p,q) \xrightarrow{a,*} (p',q) \mid p \xrightarrow{a} p' \in P\} \\
&\cup \{(p,q) \xrightarrow{*,b} (p,q') \mid q \xrightarrow{a} q' \in Q\}
\end{aligned}$$

which is as one would expect for the composition of asynchronous processes (a'la CCS [27]): the components can either evolve together through visible communication, or can evolve independently through internal actions.

3.3.2 Weak bisimulation

We now turn to obtaining a cover system for weak bisimulation, and note that the list monad in **Set** gives rise to the path monad on transition systems: for A an object of Tran(**Set**) define $MA \stackrel{def}{=} (i \in S,\ \bigcup_{i<\omega} R^i)$, where

$$\begin{aligned} R^0 &= \{(s, [], s) \mid s \in S\} \\ R^{i+1} &= \{(s, a :: \ell, t) \mid \exists u.\ (s, a, u) \in R \wedge (u, \ell, t) \in R^i\} \end{aligned}$$

For transition system A, the transitions of MA are all the finite sequences of transitions in A.

Proposition 3.3.3 *M is a stable functor on* Tran(**Set**).

Proof. MA is a deterministic transition system as all R_i are deterministic and involve distinct labels. The effect of M on maps is given componentwise by the identity and list monads, and a simple induction on the structure of the labels shows this is well-defined.

To see M is stable, let P be the pullback of f and g and consider the induced map h to the pullback of Mf and Mg:

$$\begin{tikzcd} MP \arrow[rr, "M\pi"] \arrow[dd, "M\pi'"'] \arrow[dr, dashed, "h"] & & MA \arrow[dd, "Mf"] \\ & Q \arrow[ur, "\pi"] \arrow[dl, "\pi'"] & \\ MB \arrow[rr, "Mg"'] & & MC \end{tikzcd}$$

Define $h' : Q \to MP$ such that $(s, t) \mapsto (s, t)$ and $(l, m) \mapsto zip(l, m)$. An induction on the structure of the labels of Q shows h' is well-defined. To see that h' is the inverse of h it is sufficient to consider the label component and note that $unzip; zip$ is the identity on $(A{\times}B)^*$, and $zip; unzip$ is the identity on $\{(\ell, m) \mid length(\ell) = length(m)\} \subseteq A^*{\times}B^*$. □

The action θ induces (componentwise) an action strip for the path functor over the delay monad, and thus a stable functor $M_{\mathrm{strip}} : \mathrm{Tran}(\mathbf{Set})_D \to \mathrm{Tran}(\mathbf{Set})$ we will use to obtain a cover system.

Example 3.3.4 strip : *$MD \Rightarrow D$ is a stable distribution.*

Proof. One sees that strip is a morphism of transition systems by indiction on the structure of the labels of MDA. Suppose $(s,\ell,t) \in MDA$:

i) $\ell = []$ implies $s = t$ and $(s, \theta(\ell) = [], t) \in MA$.

ii) $\ell = a \operatorname{cons} m$ implies there exists $u \in S_A$ such that $(s,a,u) \in DA$ and $(u,m,t) \in MDA$. Furthermore, $(u, \theta(m), t) \in MA$ by inductive hypothesis. If $a \in \Sigma_A$ then $(s,a,u) \in A$ and so $(s, \theta(\ell) = a \operatorname{cons} \theta(m), t) \in MA$. Otherwise $u = s$ and so $(s, \theta(\ell) = \theta(m), t) \in MA$.

The fact that strip is a stable distribution is given by it's construction componentwise from the identity and θ. □

Let $\exists^{\mathcal{E}}_{\theta}$ be the cover system $M_{\theta}^{-1}(\exists^{\mathcal{E}})$. To see how the induced equivalence on processes corresponds to weak bisimulation, note that any span $A \rightsquigarrow B$ is equivalently specified as a map $P \to A{\times}B$ of Tran_D. Two such spans $f : P \to A{\times}B$ and $g : Q \to A{\times}B$ are weakly bisimilar in the presence of a symmetric relation S on the states of P and Q for which $(p,q) \in S$ and $p \xrightarrow{\ell} p' \in MP$ implies that there exists q' and m such that $q \xrightarrow{m} q' \in MQ$ and $g(\theta(m)) = f(\theta(\ell))$.

Proposition 3.3.5 *Two spans $A \rightsquigarrow B$ are $\exists^{\mathcal{E}}_{\theta}$-bisimilar if and only if they are weakly bisimilar.*

Proof. The implication is easy. To see the converse, suppose $\mathcal{S}$ is a weak bisimulation of the spans. Form a transition system R whose states are $\{(p,q) \in \mathcal{S} \,|\, f(p) = g(q)\}$, labels are $\{(x,y) | f(x) = g(y)\}$ and transitions are $\{(p,q) \xrightarrow{(x,y)} (p',q') \mid p \xrightarrow{x} p' \wedge q \xrightarrow{y} q'\}$. Note that R is a subobject of the pullback of f and g which may ignore unreachable states. It is straightforward to shown that the projections from R are $\exists^{\mathcal{E}}_{\theta}$-maps. □

Note that the formulation of weak bisimulation in this setting corresponds very closely to the first definition given by Milner in [26] rather than the description (given there as proposition 8.4) which has now become standard [27]. Note also that the functor M_{θ} also induces a cover system for weak trace equivalence by post-composing the stable functor from transition systems to trees.

3.3.3 The structure of asynchronous processes

Proposition 3.3.6 ASProc *is compact-closed.*

Proof. Applying D to the initial transition system results in the final transition system:

$$\begin{tikzcd}[column sep=small] & 0+1 \arrow[dl, "\sim"'] \arrow[dr, "\sim"] & & \\ 1\times(0+1) & & 1 & 1 \arrow[l, "="'] \end{tikzcd}$$

The result is then given by proposition 2.4.1. $\square$

Note that the product in $\mathrm{Tran}(\mathbf{Set})_D$ is not the result of lifting the product from $\mathrm{Tran}(\mathbf{Set})$. Given transition systems A and B, their product in $\mathrm{Tran}(\mathbf{Set})_D$ is given by the transition system

$$\begin{aligned} &\{(s,t)\xrightarrow{x,y}(s',t') \mid s\xrightarrow{x}s',\ t\xrightarrow{y}t'\} \\ \cup\ &\{(s,t)\xrightarrow{x,*}(s',t) \mid s\xrightarrow{x}s'\} \\ \cup\ &\{(s,t)\xrightarrow{*,y}(s,t') \mid t\xrightarrow{y}t'\} \end{aligned}$$

allowing transitions by both components simultaneously as well as independent transitions by either component.

The fact that coproducts in $\mathbf{X}$ are not stable in $\mathbf{X}_{\_+1}$ (see section 3.2.3) implies that the coproduct of transition systems does not lift to a stable or even $\exists^{\mathcal{R}}$ stable coproduct of transition systems with delay. Consequently one does not have biproducts in $\mathrm{ASProc}(\mathbf{X})$.

Finally, note that M_θ induces a faithful functor $\mathrm{Proc}(M_\theta) : \mathrm{ASProc} \to \mathrm{SProc}$ between process categories. Furthermore, since M_θ preserves finite limits $\mathrm{Proc}(M_\theta)$ preserves the tensor product.

3.4 Asynchronous games

We now consider adding delay to the models of games in sections 2.5.4 and 2.5.3. Obtaining a suitable asynchronous model category in these settings is more delicate and requires adding extra structure to the synchronous model category.

3.4.1 Interleaved games

For the model category Game of games which interleave the actions of player and opponent, one obtains a notion of delay in the following way: add a new action '*' to each alphabet, add a copy of every state to the opposite state space, and add idle transitions between the original states to their duplicates:

$$\begin{tikzcd} s^* \arrow[d, bend right, "*"'] & t^* \arrow[d, bend right, "*"'] \\ s \arrow[u, bend right, "*"'] \arrow[r, "a"] & t \arrow[u, bend right, "*"'] \end{tikzcd}$$

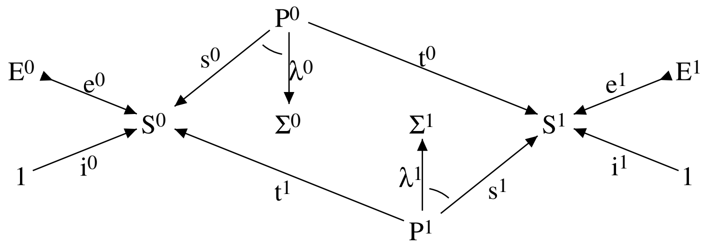
There is a problem, however, if the states introduced by delay have the same status as the original states: in escence, the monad

$$(S^0,\ S^1)\ \mapsto\ (S^0{+}S^1,\ S^0{+}S^1)$$

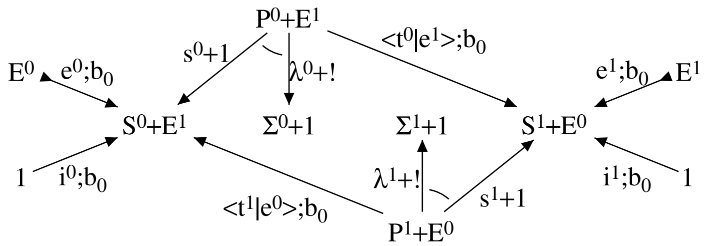
on $\mathbf{X}\times\mathbf{X}$ with unit (b_0, b_1) and multiplication (∇, ∇) is not stable. Consequently the suggested monad does not admit the construction of a process category.

The problem is overcome by distinguishing certain states of a game as being *external*. We thus modify the sketch of example 2.5.1 as follows:

Definition 3.4.1 EGame($\mathbf{X}$) *is the category of models in* $\mathbf{X}$ *of the sketch:*



The delay monad D on EGame($\mathbf{X}$) is then defined as follows:



The monad adds idling only to external states, and the states introduced by the monad are themselves non-external. Like the delay monad on transition systems, the unit and multiplication are given componentwise by the identity and $\_ + X$.

Proposition 3.4.2 *The Kleisli category* EGame$(\mathbf{X})_D$ *has pullbacks.*

Proof. It is easily shown that D is a functor and that η and μ are morphisms. The requirements of being a stable monad are then given componentwise. $\square$

To obtain cover systems for span construction we use the underlying functor to lift the cover systems from the synchronous model category. First let $\mathcal{L}_0$ be the class of morphisms in EGame($\mathbf{X}$) which are cartesian for both s^0 and e^0, and let $\mathcal{L}_1$ the morphisms cartesian for s^1 and e^1. As our category of "asynchronous" strategies we take the full subcategory of

$$\mathrm{Proc}^{U_D^{\dashv}(\mathcal{L}_0), U_D^{\dashv}(\mathcal{L}_1)}(\mathrm{EGame}(\mathbf{X}))$$

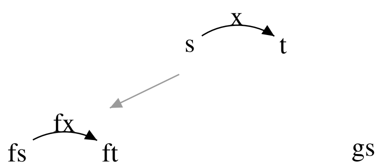
whose objects have only external states. Thus the non-external (or internal) states are explicitly used only in the implementation of processes.

To develop intuition for the behavior of these processes, consider a process $A \to B$ seen as a span in Game(**Set**):

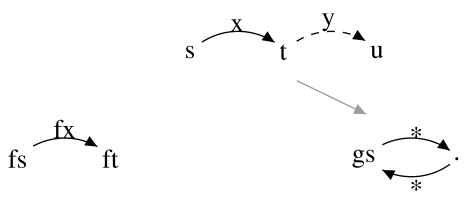
$$\begin{tikzcd} & P \arrow[dl, "f"'] \arrow[dr, "g"] & \\ TA & & TB \end{tikzcd}$$

Suppose that s is a player state in P and that fs is external in A. Since Uf reflects external states, s must also be external. Furthermore, the fact that Uf uniquely reflects player transitions means not only that s admits in P all transitions permitted at fs in A, but that s admits no transition in P corresponding to the idle transition from fs in DA — that idle transition is uniquely covered by the idle transition from s in DP. Of course the external opponent states of P stand in the same relation to the opponent states of B.

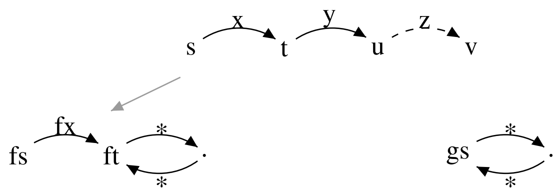
Now consider the effect of performing a transition from player state s in P which has a visible (non-idling) effect in A.



If the transition x is visible in B then t is an external opponent state at which P awaits input in B. Otherwise, x corresponds to idling in DB and (since g preserves external states) t is a non-external state which (since Ug uniquely reflects opponent transitions in DB) admits a unique transition y



corresponding to the idle-back transition from gs in DB. Continuing this line of reasoning another step: If the transition y is visible in A then u is an external player state at which P awaits input in B. Otherwise, y corresponds to idling in DA and u is a non-external state which admits a unique transition z

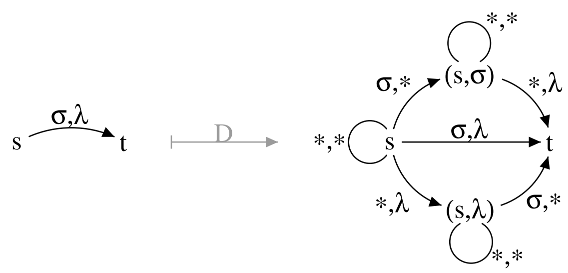


corresponding to the idle-back transition from fu in DA. The behavior of these processes should now be apparent: When in an external state, a process is reacting to the environment at the covered interface. Once a process initiates idling (as output at the non-covered interface) it's evolution is determined to either perform internal actions "forever" or to return eventually to an external state.

Note that the identity processes are still "synchronous wires", which are the identites in the category of synchronous strategies of section 2.5.4. Furthermore, the cover systems for span construction (being left-factor closed) prevent us from obtaining a cover system for equivalence which ignores any significant amount of idling. We can, however, construct a functor into ASProc which is a lifting over the respective monads of the underlying functor from games to transition systems. Quotienting by this functor turns the "asynchronous wires" (the result of breaking the synchrony in the identity processes) into idempotents. We conjecture that the result of splitting these idempotents is Abramsky and Jagadeesan's category of games and strategies [3], although not restricted to the winning strategies.

3.4.2 Simultaneous Games

Consider the games of example 2.5.2, in which both players move simultaneously. If we provide both players the ability to delay then, to maintain independence, the monad must account for the possibility that one player specifies an action while the other delays. Since the state change is not determined until both actions are given, the specified action must be encoded in the state space. The monad thus has the following effect on a transition:



where, for simplicity, we assume each player has a single permissible action have and have left the transitions from t implicit.

As in the previous example, we must add some additional structure to the synchronous model category to obtain a stable monad. In this case we add to the sketch 2.5.2 a specification of the external states e and of the external actions x^Σ and x^Λ:

$$
\begin{tikzcd}
 & \mathrm{Q} \arrow[dl, "q^\Lambda"'] \arrow[rr, "q^\Sigma"] \arrow[dd, dashed, "q"] & & \mathrm{Q}^\Sigma \arrow[dl, "y^\Sigma"'] \arrow[dd, "x^\Sigma"] \\
\mathrm{Q}^\Lambda \arrow[rr, "y^\Lambda"'] \arrow[dd, "x^\Lambda"'] & & \mathrm{E} \arrow[dd, "e"] & \\
 & \mathrm{P} \arrow[dl, "p^\Lambda"'] \arrow[rr, "p^\Sigma"] & & \mathrm{P}^\Sigma \arrow[dl, "s^\Sigma"] \\
\mathrm{P}^\Lambda \arrow[rr, "s^\Lambda"'] & & \mathrm{S} &
\end{tikzcd}
$$

Then, we define the delay monad D to leave the specified external structure unchanged and to have the following effect on the underlying structure of a game:

$$
\begin{tikzcd}
 & & \mathrm{P+Q^\Lambda+(Q^\Sigma+E)+(Q+Q^\Sigma)+(Q+Q^\Lambda)} \arrow[dl, "P_D^\Sigma"'] \arrow[dr, "P_D^\Lambda"] \arrow[rr, "t_D"] & & \mathrm{S+Q^\Sigma+Q^\Lambda} \\
 & \mathrm{P^\Sigma+E+Q^\Sigma+(Q+Q^\Lambda)} \arrow[dl, "n_D^\Sigma"'] \arrow[dr, "s_D^\Sigma"] & & \mathrm{P^\Lambda+S+(Q+Q^\Sigma)+Q^\Lambda} \arrow[dl, "s_D^\Lambda"'] \arrow[dr, "n_D^\Lambda"] & \\
\Sigma+1 & & \mathrm{S+Q^\Sigma+Q^\Lambda} & & \Lambda+1
\end{tikzcd}
$$

where we use the following definitions for the arrows in the sketch.

$$
\begin{aligned}
P_D^\Sigma &\stackrel{def}{=} \langle p^\Sigma{+}y^\Lambda | x^\Sigma{+}1\rangle + \langle q^\Sigma|1\rangle + 1 \\
P_D^\Lambda &\stackrel{def}{=} \langle p^\Lambda | x^\Lambda\rangle + \langle y^\Sigma|1\rangle + 1 + \langle q^\lambda|1\rangle \\
s_D^\Sigma &\stackrel{def}{=} \langle s^\Sigma|e\rangle + 1 + \langle q^\Lambda|1\rangle \\
s_D^\Lambda &\stackrel{def}{=} \langle s^\Lambda|e\rangle + \langle q^\Sigma|1\rangle + 1 \\
n_D^\Sigma &\stackrel{def}{=} \langle n^\Sigma{+}\,! \mid x^\Sigma; n^\Sigma; b_0 \mid q^\Sigma; x^\Sigma; n^\Sigma{+}!\rangle \\
n_D^\Lambda &\stackrel{def}{=} \langle n^\Lambda{+}\,! \mid q^\Lambda; x^\Lambda; n^\Lambda{+}\,! \mid x^\Lambda; n^\Lambda; b_0\rangle \\
t_D &\stackrel{def}{=} \langle t; b_0 \mid 1; b_2 \mid \langle 1; b_1 \mid e; b_0\rangle \mid t{+}1; b_0 \mid t{+}1; b_0{+}1\rangle
\end{aligned}
$$

The unit and multiplication of D are given componentwise by the identity and exception monads.

Proposition 3.4.3 *For lextensive* $\mathbf{X}$*, the delay monad* D *on* SGame($\mathbf{X}$) *is a stable monad.*

As in the previous example, we lift the cover systems for span construction through the underlying functor of the monad: taking $\mathcal{L}^\Sigma$ to the class of morphisms of Game($\mathbf{X}$) which are cartesian for both s^Σ and e and $\mathcal{L}^\Lambda$ to be the morphisms cartesian for both s^Λ and e, we take the full subcategory of

$$\mathrm{Proc}^{U_D^{\dashv}(\mathcal{L}^\Sigma), U_D^{\dashv}(\mathcal{L}^\Lambda)}(\mathrm{SGame}(\mathbf{X}))$$

whose objects have all states and actions external (i.e. e, x^Σ and x^Λ are identities).

Again the cover systems serve to keep processes deterministic and free of deadlock — although they are susceptible to livelock. Since the legs of a span must uniquely reflect delay on input, a process cannot perform an action until the environment provides an input. Once an input is provided the process may delay providing an output (i.e. do an internal transition based on the input), but if it does then it must deterministically perform internal actions ad infinitum or eventually perform a visible output and thus enter in a state in which it is again awaiting input.

Unfortunately the Kleisli category SGame$(\mathbf{X})_D$ does not have a final object, and thus one does not immediately obtain a tensor corresponding to the product. However, the fact that processes can delay on output raises hopes of defining the complementation map (unattainable in the synchronous setting) neccessary to obtain $*$-autonomy. Of course there is also the problem of attaining a cover system for equivalence which ignores idling, which we will not discuss here.

3.5 Summary

We have presented an approach to adding asynchrony in the construction of process categories: specifically, one uses as a model category the Kleisli category of a suitable delay monad. As Kleisli categories do not generally have pullbacks, the isolation of a class of monads which admit the construction of processes constitutes the main technical result of the chapter.

To illustrate our approach, we use the monad of delay on transition systems to reconstruct Abramsky's category ASProc of asynchronous processes modulo weak bisimulation. Here the cover system for weak bisimulation is given by an action for the delay monad — a description which bares close resemblance to Milner's first definition of weak bisimulation in [26]. We have also demonstated a delay monad on the category of games and have suggested that the resulting process category forms a basis from which to obtain Abramsky and Jagadeesan's category of games and strategies.

Note that although the delay monad exists in SProc, one does not obtain ASProc as a Kleisli construction upon SProc. This avenue is explored in Simon Gay's thesis, where it is shown that the tensor (for parallel composition) in SProc does not have a natural distribution over the delay monad.

Chapter 4

Equivalence of process categories

In this chapter we show how adjunctions between model categories can induce equivalence of the associated process categories. This provides a degree of freedom when working with a process category, as one form of model may lend itself more readily to a certain task (e.g. the treatment of biproducts in section 2.4.2).

In [33], relationships are established between many standard models of concurrencty: the linear-time models are shown to be reflective subcategories of the corresponding branching-time models, while the behavior models form coreflective subcategories of the corresponding system models. For the models of interleaved concurrency one has the following square of adjunctions,

$$\begin{tikzcd}
\text{Trace} \arrow[r, tail] \arrow[d, tail] & \text{STree} \arrow[d, tail] \\
\text{Tran} \arrow[r, tail] & \text{NTran}
\end{tikzcd}$$

where horizontal arrows are reflections and vertical arrows are coreflections. We show that these relationships pass through the process construction and in fact become adjoint equivalences: with respect to bisimulation, each of these model categories give rise to the process category SProc.

We develop a general notion behavior object in a model category, and show that when a model category has enough behaviors it is sufficient to consider only the behavior objects when constructing a process category.

4.1 Right-closure and reachable states

Here we note that the process equivalence induced by a cover system coincides with that induced by its right closure.

Proposition 4.1.1 *If $\mathcal{L}_0$, $\mathcal{L}_1$ and $\mathcal{X}$ are cover systems on $\mathbf{X}$, then $\mathrm{Proc}^{\mathcal{L}_0,\mathcal{L}_1}(\mathbf{X},\mathcal{X})$ is equivalent to $\mathrm{Proc}^{\mathcal{L}_0,\mathcal{L}_1}(\mathbf{X},\mathcal{X}{\downarrow})$.*

Proof. Any $\mathcal{X}$-bisimulation is an $\mathcal{X}{\downarrow}$-bisimulation as $\mathcal{X} \subseteq \mathcal{X}{\downarrow}$. Conversely, given an $\mathcal{X}{\downarrow}$-bisimulation (viz. z_1 and z_2)

$$\begin{tikzcd}
& & \cdot \arrow[dl] \arrow[dr] \arrow[dd, phantom, "\lrcorner" rotate=-45, near start] & & \\
& \cdot \arrow[dl] \arrow[dr, "x_1"] & & \cdot \arrow[dl, "x_2"'] \arrow[dr] & \\
\cdot & & \cdot \arrow[ll, "z_1"] \arrow[rr, "z_2"'] & & \cdot
\end{tikzcd}$$

one obtains an $\mathcal{X}$-bisimulation by pulling back the witnesses (viz. x_1 and x_2). $\square$

Note that in the category of transition systems the cover system $\exists^{\mathcal{R}}$ for bisimulation is not right-factor closed due to the presence of unreachable states However, using lemma 2.2.4, the cover system for bisimulation in the category of trees is right-factor closed since cover morphisms are retractions.

4.2 Moving between model categories

We now show that adjunctions between model categories can induce equivalence of the associated process categories. The key observation is that certain cover morphisms in a model category become isomorphisms in it's process category.

Lemma 4.2.1 *For $\mathcal{X}$ a left-factor closed cover system on $\mathbf{X}$: if $f : A \to B$ is in $\mathcal{X}$ then the process $(1, f) : A \rightsquigarrow B$ is an isomorphism in $\mathrm{Proc}(\mathbf{X},\mathcal{X})$.*

Proof. Noting left-factor closure of $\mathcal{X}$ in the rightmost diagram, the required witnesses are as follows:

$$\begin{tikzcd}
& \cdot \arrow[dl, bend right, "f"'] \arrow[d, "f"] \arrow[dr, bend left, "f"] & \\
B \arrow[r, dash, "="'] & \cdot & B \arrow[l, dash, "="]
\end{tikzcd}
\qquad
\begin{tikzcd}
& & \cdot \arrow[d, dashed] \arrow[dl, bend right, "="'] \arrow[dr, bend left, "="] & & \\
& A \arrow[dl, dash, "="'] \arrow[dr, "f"] & \cdot \arrow[l] \arrow[r] & A \arrow[dl, "f"'] \arrow[dr, dash, "="] & \\
A & & B & & A
\end{tikzcd}$$

$\square$

Consequently, a natural transformation whose components belong to a left-factor closed cover system becomes a natural isomorphism in a process category quotiented by any more generous equivalence. Let $\mathrm{Proc}(\alpha) : \mathrm{Proc}(F) \Rightarrow \mathrm{Proc}(G) : \mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X},\mathcal{X}) \to \mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y},\mathcal{Y})$.

Corollary 4.2.2 *If $\alpha : F \Rightarrow G$ is componentwise in $\mathcal{Y}_L$, with $\mathcal{Y}_L$ a left-factor closed cover system and $\mathcal{Y}_L \subseteq \mathcal{Y}$, then* $\mathrm{Proc}(\alpha)$ *is a natural isomorphism.*

For example, as $\exists^{\mathcal{I}}$ is always contained in $\exists^{\mathcal{X}}$, an adjunction between categories of "transition systems" whose unit and counit are componentwise $\exists^{\mathcal{I}}$ will give an equivalence on the associated process categories.

4.2.1 Separating initial states

Recall the category $\mathrm{STran}(\mathbf{X})$ of transition systems with separated initial states of section 2.4.2. Given an object B in $\mathrm{STran}(\mathbf{X})$ we obtain an object FB of $\mathrm{Tran}(\mathbf{X})$ by adding the initial state to the rest of the state space:

$$\begin{tikzcd}
 & & P+P^0 \arrow[dl, "m+m^0"'] \arrow[ddr, "\langle\alpha\,|\,\alpha^0\rangle;b_0"] & & \\
 & S\times\Sigma+1\times\Sigma \arrow[dl, "d"'] & & & \\
(S+1)\times\Sigma & & & S+1 & 1 \arrow[l, "b_0"]
\end{tikzcd}$$

Conversely, given an object A of $\mathrm{Tran}(\mathbf{X})$ we can extract the initial transitions and thereby form an object GA of $\mathrm{STran}(\mathbf{X})$:

$$\begin{tikzcd}
1\times\Sigma \arrow[d, "i\times 1"'] & P^0 \arrow[l, "m^0"'] \arrow[d, "x"] \arrow[dr, dashed, "\alpha^0"] \arrow[dl, phantom, very near start, "\llcorner"] & \\
S\times\Sigma & P \arrow[l, "m"] \arrow[r, "\alpha"'] & S
\end{tikzcd}$$

These functors determine an adjunction $F \dashv G : \mathrm{STran}(\mathbf{X}) \to \mathrm{Tran}(\mathbf{X})$. In $\mathrm{STran}(\mathbf{X})$, GFB differs from B in that it contains an unreachable copy of the initial state, and thus the unit η injects B into GFB. In $\mathrm{Tran}(\mathbf{X})$, FGA differs from A in that it has a new initial state which is bisimilar to the old (still reachable) initial state, so the counit ϵ_A collapses the new initial state onto the original

For $\mathbf{X}$ a lextensive category with cover system $\mathcal{X}$,

Proposition 4.2.3 $\mathrm{Proc}(\mathrm{STran}(\mathbf{X}), \exists^{\mathcal{X}})$ *is equivalent to* $\mathrm{Proc}(\mathrm{Tran}(\mathbf{X}), \exists^{\mathcal{X}})$ *whenever* $+$ *preserves* $\mathcal{X}$.

Proof. F and G are easily seen to be functors, and are stable componentwise. So to be $\exists^{\mathcal{X}}$-stable it is sufficient to preserve $\exists^{\mathcal{X}}$. Concerning F, note that the arrow required to be $\mathcal{X}$-cartesian in FB is the sum $(m^0; p_0) + (m; p_0)$. To see G preserves $\exists^{\mathcal{X}}$, recall that the arrows required to be $\mathcal{X}$-cartesian in GA are $m; p_0$ and $m^0; p_0$: the former is given and the latter is the pullback of $m; p_0$ along i.

Noting that the initial transitions of FB are extracted as

$$\begin{tikzcd}
1\times\Sigma \arrow[d, "b_1\times 1"'] & 1\times\Sigma \arrow[l, "="'] \arrow[d, "b_1"] \arrow[dl, phantom, "\lrcorner"] & P^0 \arrow[l, "m^0"'] \arrow[d, "b_1"] \arrow[dl, phantom, "\lrcorner"] \arrow[dr, "\alpha^0"] & \\
(S+1)\times\Sigma & S\times\Sigma+1\times\Sigma \arrow[l, "d"] & P+P^0 \arrow[l, "m+m^0"] \arrow[r, "\langle\alpha|\alpha^0\rangle"'] & S
\end{tikzcd}$$

the unit η_B is given by:

$$\begin{tikzcd}
1\times\Sigma \arrow[d, "="'] & P^0 \arrow[l, "m^0"'] \arrow[r, "\alpha^0"] \arrow[d, "="] & S \arrow[d, "b_0"] & P \arrow[l, "\alpha"'] \arrow[r, "m"] \arrow[d, "b_0"] & S\times\Sigma \arrow[d, "b_0"] \\
1\times\Sigma & P^0 \arrow[l, "m^0"] \arrow[r, "\alpha^0;b_0"'] & S+1 & P+P^0 \arrow[l, "\langle\alpha|\alpha^0\rangle;b_0"] \arrow[r, "m+m^0;d"'] & (S+1)\times\Sigma
\end{tikzcd}$$

This morphism belongs to $\exists^{\mathcal{I}}$ since it is the identity on initial transitions and b_0 is cartesian.

The counit is as follows

$$\begin{tikzcd}
(S+1)\times\Sigma \arrow[d, "\langle 1|i\rangle\times 1"'] & P+P^0 \arrow[l, "m+m^0;d"'] \arrow[r, "\langle\alpha|\alpha^0\rangle;b_0"] \arrow[d, "\langle 1|x\rangle"'] & S+1 \arrow[d, "\langle 1|i\rangle"'] & 1 \arrow[l, "b_1"'] \arrow[d, "="] \\
S\times\Sigma & P \arrow[l, "m"] \arrow[r, "\alpha"'] & S & 1 \arrow[l, "i"]
\end{tikzcd}$$

This morphism is in $\exists^{\mathcal{I}}$ also as coproducts are stable and ∇ is cartesian.

Thus the adjunction lies in the domain of Proc and becomes an equivalence of the associated process categories. $\square$

Although the move from transition systems to transition systems with separated initial states made it easier to obtain biproducts, it actually had no effect on the resulting process category.

4.2.2 Nondeterministic transition systems

Consider the category of NTran($\mathbf{X}$) of nondeterministic transition systems given by the following sketch:

$$\begin{tikzcd}
 & P \arrow[dl, "m"'] \arrow[dr, "\alpha"] & & \\
S\times\Sigma & & S & 1 \arrow[l, "i"']
\end{tikzcd}$$

These transition systems differ from their deterministic counterparts only in that there may be many transitions with the same source state and label (viz. m is not monic).

A nondeterministic transition system A gives rise to a deterministic transition system FA by adding state information to the labels:

$$\begin{tikzcd}
& P \arrow[dl, "\langle m;p_0,1\rangle"'] \arrow[dr, "\alpha"] & & \\
S\times P & & S & 1 \arrow[l, "i"]
\end{tikzcd}$$

Note that $\langle f, 1\rangle$ is always monic, and F preserves $\exists^{\mathcal{X}}$ since $\langle m;p_0,1\rangle;p_0 = m;p_0$. Although this functor is not an adjoint to the inclusion functor $I : \mathrm{Tran}(\mathbf{X}) \to \mathrm{NTran}(\mathbf{X})$, the morphism

$$\begin{tikzcd}
S\times P \arrow[d, "1\times(m;p_1)"'] & P \arrow[l, "\langle m;p_0,1\rangle"'] \arrow[r, "\alpha"] \arrow[d, "="] & S \arrow[d, "="] & 1 \arrow[l, "i"'] \arrow[d, "="] \\
S\times\Sigma & P \arrow[l, "m"] \arrow[r, "\alpha"'] & S & 1 \arrow[l, "i"]
\end{tikzcd}$$

serves both as a natural transformation $IFA \Rightarrow A$ and $FIB \Rightarrow B$. Furthermore, it is pointwise in $\exists^{\mathcal{I}}$ (as it has identity effect on states and transitions) and consequently:

Proposition 4.2.4 $\mathrm{Proc}(\mathrm{NTran}(\mathbf{X}), \exists^{\mathcal{X}})$ *is equivalent to* $\mathrm{Proc}(\mathrm{Tran}(\mathbf{X}), \exists^{\mathcal{X}})$

This result suggests that accounting explicitly for nondeterminism in a model category is not necessary when constructing a process category.

Note we have not actually used the categories of deterministic and nondeterministic transition systems described by [33]: the latter are Kleisli categories of appropriately defined *delay* monads upon $\mathrm{Tran}(\mathbf{Set})$ and (a slight modification of) $\mathrm{NTran}(\mathbf{Set})$, respectively.

4.2.3 Unlabelled transition systems

One obtains an equivalent process category beginning even with unlabelled transition systems. Consider the category $\mathrm{UTran}(\mathbf{X})$ of given by the following sketch:

$$\begin{tikzcd}
& & P \arrow[dl, "s"'] \arrow[dr, "t"] & \\
1 \arrow[r, "i"] & S & & S
\end{tikzcd}$$

In this case there is an adjunction $\mathrm{Tran}(\mathbf{X}) \dashv \mathrm{UTran}(\mathbf{X})$: the right adjoint discards labels, and the left adjoint uses the permission set to introduce labels. The unit is the identity and the counit at object A is the label component of A (viz. $m;p_1$ in the sketch Tran) — both belonging to $\exists^{\mathcal{I}}$.

Proposition 4.2.5 Proc(UTran($\mathbf{X}$), $\exists^{\mathcal{X}}$) *is equivalent to* Proc(Tran($\mathbf{X}$), $\exists^{\mathcal{X}}$).

This suggest that explicitly labelling of transitions in a model category has no effect on the resulting process category.

4.3 Behavior models

Here we provide an abstract notion of behavior and show that, for an appropriate cover system, it is sufficient to consider the category of behaviors when building a process category. We further link the existence of such subcategories of behaviors to the existence of inductive datatypes in span categories.

4.3.1 Abstract behaviors

Let $\mathcal{X}$ be a left-factor closed cover system on $\mathbf{C}$:

Definition 4.3.1 *An object Z of $\mathbf{C}$ is an $\mathcal{X}$-behavior if given any $x : A \to B$ in $\mathcal{X}$, every $f : Z \to B$ factors uniquely through x:*

$$\begin{tikzcd} A \arrow[d, "x"'] & Z \arrow[l, dashed] \arrow[dl, "f"] \\ B & \end{tikzcd}$$

Intuitively, an $\mathcal{X}$-behavior cannot distinguish objects which are $\mathcal{X}$-equivalent. Note that any $\mathcal{X}$-morphism $x : Z_1 \to Z_2$ between $\mathcal{X}$-behaviors is an isomorphism: x has a left inverse x'

$$\begin{tikzcd} Z_1 \arrow[d, "x"'] & Z_2 \arrow[l, dashed, "x'"'] \arrow[dl, "="] \\ Z_2 & \end{tikzcd}$$

which (by left-factor closure) belongs to $\mathcal{X}$. Similarly x' has a left-inverse and is thus an isomorphism.

If Z is an $\mathcal{X}$-behavior and $Z \to A$ an $\mathcal{X}$-morphism then we regard Z as an *$\mathcal{X}$-behavior of A*. By left-factor closure and the preceeding observation we see that all $\mathcal{X}$-behaviors for a given object are isomorphic:

$$\begin{tikzcd} Z_1 \arrow[d, "x_1"'] & Z_2 \arrow[l, dashed] \arrow[dl, "x_2"] \\ A & \end{tikzcd}$$

A category $\mathbf{C}$ is said to have *enough $\mathcal{X}$-behaviors* if for every object A there is an $\mathcal{X}$-behavior TA and a morphism $\epsilon : TA \to A$ in $\mathcal{X}$.

Proposition 4.3.2 *If $\mathbf{C}$ has enough $\mathcal{X}$-behaviors then T is a coreflective subcategory with counit ϵ.*

Proof. The couniversal diagram is an instance of the defining property of $\mathcal{X}$-behavior:

$$\begin{tikzcd}
A & ITA \arrow[l, "\varepsilon"'] \\
 & IZ \arrow[ul, "f"] \arrow[u, dashed]
\end{tikzcd}$$

where I is the inclusion functor. □

For any left-factor closed cover system $covX$ in $\mathbf{C}$, if $\mathbf{C}$ has enough $\mathcal{X}$-behaviors then, taking the equivalence to be any cover system containing $\mathcal{X}$, the process categories constructed either of $\mathbf{C}$ or the full coreflective subcategory $\mathbf{C}^T$ coincide. Writing $\mathcal{Y}^T$ for the restriction of $\mathcal{Y}$ to $\mathbf{C}^T$:

Proposition 4.3.3 *If $\mathcal{X} \subseteq \mathcal{Y}$ and T preserves $\mathcal{Y}$ then* $\mathrm{Proc}(\mathbf{C}, \mathcal{Y})$ *is equivalent to* $\mathrm{Proc}(\mathbf{C}^T, \mathcal{Y}^T)$.

Proof. T is $\mathcal{Y}$-stable since it is stable (it is a right adjoint) and preserves $\mathcal{Y}$. As the counit ϵ is pointwise $\mathcal{X}$ it is (by lemma 2.3.8) $\mathcal{X}$-cartesian and thus $\mathcal{Y}$-cartesian. The coreflection is thus preserved by the process construction and (by proposition 4.2.2) becomes an equivalence. □

Any coreflective subcategory T of $\mathbf{C}$ determines a cover system for which T gives enough behaviors: the class $T^{-1}(\mathcal{I})$ (the morphisms of $\mathbf{C}$ taken by T to isomorphisms) is a left-factor closed cover system and, as T is an idempotent comonad [6], the counit ϵ belongs to $T^{-1}(\mathcal{I})$. In fact,

Proposition 4.3.4 *If $\mathcal{X} \subseteq T^{-1}(\mathcal{I})$ is left-factor closed and contains ϵ then:*

i) $\mathbf{C}$ *has enough $\mathcal{X}$-behaviors;*

ii) $T^{-1}(\mathcal{I}) = \mathcal{X}{\downarrow}$.

Proof. i) Let $f : TA \to B$ and let $x : C \to B$ belong to $\mathcal{X}$. Since Tx and ϵ_T are isomorphims, there is a morphism $f' : TA \to C$ given by $\epsilon_{TA}^{-1}; Tf; Tx^{-1}; \epsilon_C$ which (by naturality) satisfies $f'; x = f$.

$$\begin{tikzcd}
TC \arrow[rr, "\varepsilon"] \arrow[d, dash, "Tx"'] & & C \arrow[d, "x"] & \\
TB \arrow[rr, "\varepsilon"] & & B & \\
 & TTA \arrow[ul, "Tf"] \arrow[rr, dash, "\varepsilon_T"'] & & TA \arrow[ul, "f"'] \arrow[uul, dashed]
\end{tikzcd}$$

If g also satisfies $f = g; x$ then $g = \epsilon_{TA}^{-1}; Tg; \epsilon_C = \epsilon_{TA}^{-1}; Tf; Tx^{-1}; \epsilon_C$ as required.

ii) If f is in $T^{-1}(\mathcal{I})$ then f is in $\mathcal{X}\!\downarrow$ since $\epsilon; f = Tf; \epsilon$ is in $\mathcal{X}$. Conversely, let x be witness that f is in $\mathcal{X}\!\downarrow$. Then Tx and $Tx; Tf$ are isomorphisms and thus Tf is an isomorphism.

$\square$

Behaviors of transition systems

A *locos* $\mathbf{C}$ is a lextensive category with list construction (see [11]): i.e. there is an endofunctor L and natural transformations $\mathrm{nil} : 1 \to LA$ and $\mathrm{cons} : LA{\times}A \to LA$ with the property that given any $f : 1 \to C$ and $g : C{\times}A \to C$ there exists a unique morphism $\mathrm{fold}_{f,g}$ such that

$$\begin{tikzcd}
LA\times A \arrow[r, "cons"] \arrow[d, "fold_{f,g}\times 1"'] & LA \arrow[d, "fold_{f,g}"'] & 1 \arrow[l, "nil"'] \arrow[dl, "f"] \\
C\times A \arrow[r, "g"'] & C &
\end{tikzcd}$$

We say that a locos $\mathbf{C}$ has *span list construction* if given any $f : 1 \to C$ and $g : C \otimes A \to C$ in $\mathrm{Span}(\mathbf{C})$ there exists r such that

$$\begin{tikzcd}
LA\otimes A \arrow[r, "cons"] \arrow[d, "r\otimes 1"'] & LA \arrow[d, "r"'] & 1 \arrow[l, "nil"'] \arrow[dl, "f"] \\
C\otimes A \arrow[r, "g"'] & C &
\end{tikzcd}$$

commutes in $\mathrm{Span}(\mathbf{C})$ and, furthermore, given $q : LA \to C$ and $\alpha : \mathrm{cons} \,;\!;\, q \Rightarrow q \otimes 1 \,;\!;\, g$ there exists a unique $\beta : q \Rightarrow r$ such that $\alpha; (\beta \otimes 1 \,;\!;\, g) = \mathrm{cons} \,;\!;\, \beta$.

Let $\mathbf{C}$ be a locos with span list construction. Each transition system A in $\mathbf{C}$ then induces the following structure in $\mathbf{C}$:

$$\begin{tikzcd}
L\Sigma_A\times\Sigma_A & L\Sigma_A\times\Sigma_A \arrow[l, "="'] \arrow[r, "cons"] & L\Sigma_A & 1 \arrow[l, "nil"'] \\
S_{TA}\times\Sigma_A \arrow[u, "\omega_S\times 1"] \arrow[d, "\varepsilon_S\times 1"'] & P_{TA} \arrow[l, "m_{TA}"'] \arrow[r, "\alpha_{TA}"'] \arrow[u, "\omega_P"] \arrow[d, "\varepsilon_P"] & S_{TA} \arrow[u, "\omega_S"] \arrow[d, "\varepsilon_S"'] & 1 \arrow[l, "i_{TA}"'] \arrow[u, "="'] \arrow[dl, "i_A"] \\
S_A\times\Sigma_A & P_A \arrow[l, "m_A"] \arrow[r, "\alpha_A"'] & S_A &
\end{tikzcd}$$

We define the transition system TA — the behavior machine of A — and the morphisms $\epsilon_A : TA \to A$ and $\omega_A : TA \to LA$ of transition systems as indicated (LA denotes the free transition system on the alphabet of A).

For a deterministic transition system in **Set**, the behavior machine is the Hoare language [19] generated by A: i.e. S_{TA} is the nonempty and prefix-closed set of strings

$$\{\, a_1 \ldots a_n \mid \exists s_1, \ldots s_n.\ i \xrightarrow{a_1} s_1 \longrightarrow \ldots \xrightarrow{a_n} s_n \,\},$$

and TA admits a transition $s \xrightarrow{a} sa$ iff $sa \in S_{TA}$. For a nondeterministic transition system, TA is the synchronization tree generated by A: i.e. the corresponding (iso-similar) transition system in which every state is reachable from the initial state by exactly one sequence of $n \geq 0$ transitions. It is clear that synchronization trees differ from traces only in that the former do not identify states with strings. Although for different morphisms, it is shown in [33] that the category of traces (resp. synchronization trees) is a coreflective subcategory of deterministic (resp. nondeterministic) transition systems.

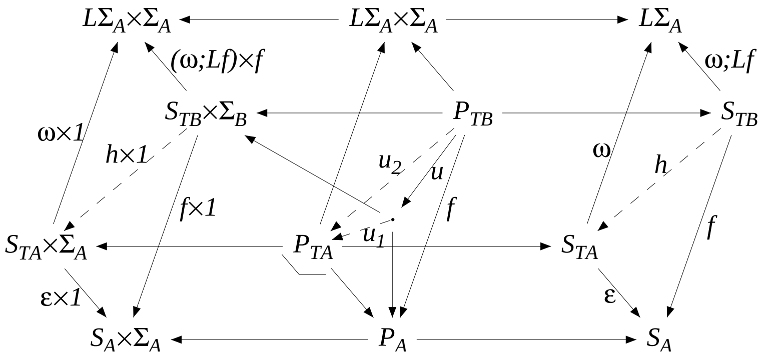
Taking Trace($\mathbf{C}$) and STree($\mathbf{C}$) to be the full subcategories of behavior machines in Tran($\mathbf{C}$) and NTran($\mathbf{C}$) respectively,

Proposition 4.3.5 Trace($\mathbf{C}$) *is a coreflective subcategory of* Tran($\mathbf{C}$), *and* STree($\mathbf{C}$) *is a coreflective subcategory of* NTran($\mathbf{C}$).

Proof. For any morphism $f : TB \to A$ of transition systems, one has in $\mathbf{C}$ the following structure:

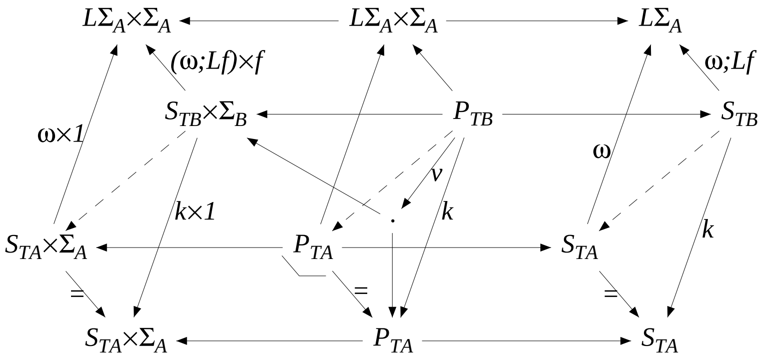
$$\begin{tikzcd}
L\Sigma_A\times\Sigma_A & L\Sigma_A\times\Sigma_A \arrow[l, "="'] \arrow[r, "cons"] & L\Sigma_A \\
L\Sigma_B\times\Sigma_B \arrow[u, "Lf\times f"] & L\Sigma_B\times\Sigma_B \arrow[l, "="'] \arrow[r, "cons"] \arrow[u] & L\Sigma_B \arrow[u, "Lf"'] \\
S_{TB}\times\Sigma_B \arrow[u, "\omega\times 1"] \arrow[d, "f\times 1"'] & P_{TB} \arrow[l, "m_{TB}"'] \arrow[r, "\alpha_{TB}"] \arrow[u] \arrow[d, "f"] \arrow[dl, phantom, "u"] & S_{TB} \arrow[u, "\omega"'] \arrow[d, "f"] \\
S_A\times\Sigma_A & P_A \arrow[l, "m_A"] \arrow[r, "\alpha_A"'] & S_A
\end{tikzcd}$$

which is a 2-cell u : cons ;; $(\omega_B; Lf, f) \Rightarrow (\omega_B{\times}1\,; Lf\times f, f{\times}f)$;; (m_A, α_A) in Span($\mathbf{C}$). By the universal property of span list construction, there exists $h : S_{TB} \to S_{TA}$ such that:



where the morphisms labelled u_1 and u_2 are the 2-cells $h{\times}1 ;; (m_A, \alpha_A)$ and $\mathrm{cons} ;; h$ respectively. This gives a morphism $h : TB \to TA$ of transition systems satisfying $h; \epsilon_A = f$.

If k is such that $k; \epsilon_A = f$ then once more by the universal property of span list construction we have the following commuting diagram in $\mathbf{C}$:

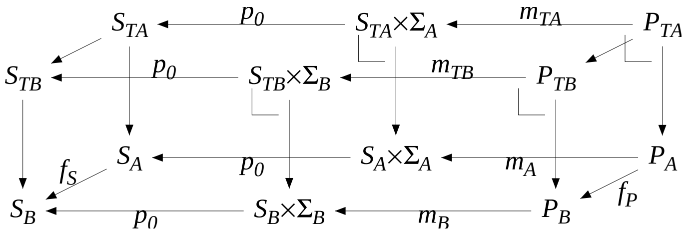


Consequently, k gives a 2-cell satisfying $u; k{\times}1 ;; (m_A, \alpha_A) = \mathrm{cons} ;; k$ which forces $k = h$. $\square$

Now, what is the cover system for which traces/synchronization trees are the behaviors of transition systems? First note that the cover systems $\exists^{\mathcal{X}}$ on behaviors are simply restrictions of the corresponding cover systems on transition systems. Then for any cover system $\mathcal{X}$ on $\mathbf{C}$,

Lemma 4.3.6 *The behavior functors T preserve $\exists^{\mathcal{X}}$.*

Proof. Any morphism $f : A \to B$ of transition systems gives the following structure in $\mathbf{C}$:



By lemma 2.2.4, if the bottom is an $\mathcal{X}$-pullback then the top will be an $\mathcal{X}$-pullback also. $\square$

The $\exists^{\mathcal{I}}$-morphisms (i.e. the local isomorphisms) of behavior machines are isomorphisms of states, but not necessarily isomorphims of labels. The (fibration) functor δ which takes a transition system to its labels, is stable and thus allows one to restrict the $\exists^{\mathcal{I}}$-morphisms of transition systems to those whose label components are isomorphisms:

Proposition 4.3.7 Tran($\mathbf{C}$) *and* NTran($\mathbf{C}$) *have* $\exists^{\mathcal{I}} \cap \delta^{-1}(\mathcal{I})$*-behaviors given by* Trace($\mathbf{C}$) *and* STree($\mathbf{C}$) *respectively.*

Proof. It is enough to see that for $f : A \to B$ in $\exists^{\mathcal{I}} \cap \delta^{-1}(\mathcal{I})$, any behavior machine of A is a behavior machine of B:

$$\begin{tikzcd}
S_{TA}\times\Sigma \arrow[d, "\varepsilon\times 1"'] & P_{TA} \arrow[l, "m_{TA}"'] \arrow[r, "\alpha_{TA}"] \arrow[d, "\varepsilon"] \arrow[dl, phantom, "\llcorner", very near start] & S_{TA} \arrow[d, "\varepsilon"] \\
S_A\times\Sigma \arrow[d, "f\times 1"'] & P_A \arrow[l, "m_A"'] \arrow[r, "\alpha_A"] \arrow[d, "f"] \arrow[dl, phantom, "\llcorner", very near start] & S_A \arrow[d, "f"] \\
S_B\times\Sigma & P_B \arrow[l, "m_B"] \arrow[r, "\alpha_B"'] & S_B
\end{tikzcd}$$

assuming for simplicity that $\Sigma_A = \Sigma_B = \Sigma$. $\square$

As $\exists^{\mathcal{I}}$ is contained in $\exists^{\mathcal{X}}$ for any $\mathcal{X}$,

Proposition 4.3.8 *For the cover systems* $\exists^{\mathcal{X}}$*, the process categories constructed upon each of the following are equivalent:* Tran($\mathbf{C}$), NTran($\mathbf{C}$), Trace($\mathbf{C}$) *and* STree($\mathbf{C}$).

In fact, one can construct an equivalent process category upon an even simpler model category. To see this, note that an equivalent presentation of synchronization trees is as labelled trees [29]: i.e. STree($\mathbf{X}$) is the category of models in $\mathbf{X}$ of the following sketch:

$$\begin{tikzcd}
1 & S_0 \arrow[l, "s_0"'] \arrow[d, "\lambda_0"] & S_1 \arrow[l, "s_1"'] \arrow[d, "\lambda_1"] & S_2 \arrow[l, "s_2"'] \arrow[d, "\lambda_2"] & \cdots \arrow[l] \\
 & \Sigma & \Sigma & \Sigma &
\end{tikzcd}$$

Similarly, the category of traces is obtained from the same sketch by requiring the labelling arrows λ_i to be monic. As in the category Tree($\mathbf{X}$) of trees in $\mathbf{X}$, the cover system $\exists^{\mathcal{X}}$ consists of the morphisms $f : A \to B$ which are $\mathcal{X}$-cartesian for all s_i.

Tree($\mathbf{C}$) is a coreflective subcategory of both STree($\mathbf{C}$) and Trace($\mathbf{X}$): the functor U which forgets the labelling is right adjoint to the (inclusion) functor

$$\begin{tikzcd} P_0 & P_1 \arrow[l] & \cdots \arrow[l] & {} \arrow[r, mapsto] & {} & P_0 \arrow[d, "b_0"'] & P_1 \arrow[l] \arrow[d, "b_1"'] & \cdots \arrow[l] \\ & & & & & \Sigma_{i\in\omega} P_i & \Sigma_{i\in\omega} P_i & \end{tikzcd}$$

which uniquely labels each transition. The counit is the identity on "states", and on labels is the copairing of the labellings of the original synchronization tree:

$$\begin{tikzcd} P_i \arrow[d, "\lambda_i"'] & P_i \arrow[l, "="'] \arrow[d, "b_i"] \\ A & \Sigma_{i\in\omega} P_i \arrow[l, "{[\lambda_i]_{i\in\omega}}"] \end{tikzcd}$$

Considering the composite coreflections between transition systems and trees, the functors preserve $\exists^{\mathcal{I}}$ and the counits are in $\exists^{\mathcal{I}}$. Thus,

Proposition 4.3.9 Tree($\mathbf{C}$) *gives* $\exists^{\mathcal{I}}$*-behaviors for* Tran($\mathbf{C}$) *and* NTran($\mathbf{C}$).

4.4 Summary

We have demonstrated that the process construction admits a degree of freedom in the choice of a model category. This provides an important link between the state-based formalisms which facilitate finite presentation of processes and the tree-based formalisms which more readily disclose the structure of a process category.

Concerning Abramsky's category of synchronous processes, we can make the following summary:

Theorem 4.4.1 *With respect to the cover system* $\exists^{\mathcal{E}}$ *for bisimulation, the process categories constructed upon each of the following model categories are equivalent to* SProc*:*

- Tran($\mathbf{Set}$)*, of deterministic transition systems;*
- NTran($\mathbf{Set}$)*, of nondeterministic transition systems;*
- UTran($\mathbf{Set}$)*, of unlabelled transition systems;*
- Trace($\mathbf{Set}$)*, of trace specifications;*
- STree($\mathbf{Set}$)*, of synchronization trees;*
- Tree($\mathbf{Set}$)*, of trees.*

This result suggests that common distinctions such as "linear vs. branching" and "system vs. behavior" between model categories do not persist to the level of process categories. It shows that the only significant feature of a model category for SProc is "extension in time".

As a final remark: since the cover system for trace equivalence in the category of trees is given by the regular epimorphisms, we can now see that trace equivalence is given in each of the other model categories via the stable functor into trees.

Chapter 5

Implementing synchronous processes

This chapter can be seen as providing a basis for programming in SProc — specifically, in the subcategories of deterministic and functional processes presented in section 2.5.1.

We begin with a category of circuits, whose objects are alphabets (sets of symbols) and whose morphisms are automata over these alphabets with the ability to halt on "unacceptable" inputs. Such automata consist of a state space S with a chosen initial state, input and output alphabets A and B, and a function $A \times S \to (S \times B) + 1$.

The category of circuits is a copy category, and the general structure therein provides notions of safety and liveness for circuits. A category of circuits which meet safety specifications is obtained by splitting the propositional idempotents, and is shown to be equivalent to the category of DSProc of deterministic processes. By extracting the strict morphisms of this category one obtains the category FSProc of functional processes.

As the functional processes are the "maps" of SProc modulo trace equivalence, the category of circuits provides a natural setting for process implementation: one expresses both circuits and specifications as elements of coalgebraic datatypes, and then establishes safety and liveness properties by proving equations involving these coalgebras — i.e. by demonstrating bisimulations.

In the remainder of this chapter, we review the relevant results from the theory of copy categories, present the construction of circuits, and finally demonstrate the correspondence between circuits and synchronous processes.

5.1 Copy categories and partial map classifiers

A *copy category* is a category with a symmetric tensor $(\otimes, \top)$ and a natural comultiplication Δ satisfying the following equations:

$$
\begin{tikzcd}
& A \arrow[dl, "\Delta"'] \arrow[dr, "\Delta"] & \\
A\otimes A \arrow[rr, "c"'] & & A\otimes A
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "\Delta"] \arrow[d, "\Delta"'] & A\otimes A \arrow[r, "\Delta\otimes 1"] & A\otimes A\otimes A \arrow[d, "a"] \\
A\otimes A \arrow[rr, "1\otimes\Delta"'] & & A\otimes(A\otimes A)
\end{tikzcd}
$$

$$
\begin{tikzcd}
& T \arrow[dl, "\Delta"'] \arrow[dr, "u^{-1}"] & \\
T\otimes T \arrow[rr, "="'] & & T\otimes T
\end{tikzcd}
\qquad
\begin{tikzcd}
& A\otimes B \arrow[dl, "\Delta\otimes\Delta"'] \arrow[dr, "\Delta"] & \\
A\otimes A\otimes(B\otimes B) \arrow[rr, "ex"'] & & A\otimes B\otimes(A\otimes B)
\end{tikzcd}
$$

Note that a cartesian category, that is a category with finite products, is a copy category in which the tensor unit is the final object.

A *copy functor* is a functor F between copy categories which is comonoidal: i.e. there exist natural transformations $\sigma_\otimes : F(A \otimes B) \to FA \otimes FB$ and $\sigma_\top : F\top \to \top$ such that

$$
\begin{tikzcd}
F(A\otimes B) \arrow[r, "\sigma"] \arrow[d, "Fc"'] & FA\otimes FB \arrow[d, "c"] \\
F(B\otimes FA) \arrow[r, "\sigma"'] & FB\otimes FA
\end{tikzcd}
\qquad
\begin{tikzcd}
F(A\otimes B\otimes C) \arrow[r, "\sigma"] \arrow[d, "Fa"'] & F(A\otimes B)\otimes FC \arrow[r, "\sigma\otimes 1"] & FA\otimes FB\otimes FC \arrow[d, "a"] \\
F(A\otimes(B\otimes C)) \arrow[r, "\sigma"'] & FA\otimes F(B\otimes C) \arrow[r, "1\otimes\sigma"'] & FA\otimes(FB\otimes FC)
\end{tikzcd}
$$

$$
\begin{tikzcd}
& FA \arrow[dl, "F\Delta"'] \arrow[dr, "\Delta"] & \\
F(A\otimes A) \arrow[rr, "\sigma"'] & & FA\otimes FA
\end{tikzcd}
\qquad
\begin{tikzcd}
F(T\otimes A) \arrow[r, "\sigma_\otimes"] \arrow[d, "Fu"'] & FA \arrow[d, "\sigma_T\otimes 1"] \\
FT\otimes FA & T\otimes FA \arrow[l, "u"]
\end{tikzcd}
$$

Similarly, a *copy transformation* is a natural transformation $\alpha : F \Rightarrow G$ between copy functors which is comonoidal: i.e.

$$
\begin{tikzcd}
FT \arrow[rr, "\alpha"] \arrow[dr, "\sigma^F"'] & & GT \arrow[dl, "\sigma^G"] \\
& T &
\end{tikzcd}
\qquad
\begin{tikzcd}
F(A\otimes B) \arrow[r, "\alpha"] \arrow[d, "\sigma^F"'] & G(A\otimes B) \arrow[d, "\sigma^G"] \\
FA\otimes FB \arrow[r, "\alpha\otimes\alpha"'] & GA\otimes GB
\end{tikzcd}
$$

Of course copy categories, copy functors and copy transformations form a 2-category **Copy**, and in this 2-category lives the embedding of circuits into processes.

The techniques for expressing safety of circuits are based upon the following observation. Any morphism $x : A \to \top$ in a copy category induces an idempotent as follows:

$$\begin{tikzcd} A \arrow[r, dashed, "e_x"] \arrow[d, "\Delta"'] & A \\ A \otimes A \arrow[r, "x \otimes 1"'] & T \otimes A \arrow[u, "u"'] \end{tikzcd}$$

Here x is called a *copy proposition*, and e_x is called a *copy idempotent*. If e_x is the identity on A then x is the *counit* for A, and is denoted $!_A$. Counits are unique if they exist, and a copy category in which every object has a counit is said to be *strict*.

Given a copy category $\mathbf{C}$, one forms a strict copy category PSplit($\mathbf{C}$) by splitting the copy idempotents. Explicitly, PSplit($\mathbf{C}$) is the category whose objects are the copy propositions x of $\mathbf{C}$, and whose morphisms $x \to y$ are those morphisms $f : \mathrm{dom}(x) \to \mathrm{dom}(y)$ of $\mathbf{C}$ for which $f = e_x; f; e_y$. The counit for an object x of PSplit($\mathbf{C}$) is given by the morphism x. The construction PSplit is a 2-functor upon $\underline{\mathbf{Copy}}$.

The method of isolating circuits which are live is based on the following notion: a morphism $f : A \to B$ of a strict copy category is called a *strict map* when $!_A = f; !_B$. If one discards all but the strict maps, the tensor unit becomes the final object and the copy tensor becomes the product. This technique of extracting the strict maps of PSplit($\mathbf{C}$) to obtain a cartesian category is called *copy completing* $\mathbf{C}$ and the resulting category is denoted CProp($\mathbf{C}$).

$\star$ reference Carboni [10], Rosellini [32], and Robinson and Rosellini [31]...

To obtain a full embedding of circuits into processes, we must use special monics when constructing transition systems. Given a cover system $\mathcal{S}$ of monics in a category $\mathbf{X}$, a partial map classifier for $\mathcal{S}$ consists of a function H on the objects of $\mathbf{X}$ together with a family $\eta_A : A \to HA$ of maps belonging to $\mathcal{S}$ with the property that for any $\mathcal{S}$-partial map (s, f) there exists a unique morphism $(s, f)^*$ in $\mathbf{X}$ for which

$$\begin{tikzcd} P \arrow[r, "f"] \arrow[d, "s"'] \arrow[dr, phantom, "\lrcorner", very near start] & B \arrow[d, "\eta_B"] \\ A \arrow[r, dashed, "(s,f)^*"'] & HB \end{tikzcd}$$

is a pullback. Mulry [?] has shown that a partial map classifier is a monoidal monad, with $H(f)$ given by $(\eta, f)^*$ and the monad multiplication given by $(1, \eta; \eta)^*$. We shall use $\mathcal{H}$ to denote a partial map classifier consisting of monad H and classified system of monics $\mathcal{M}_{\mathcal{H}}$.

We say that a partial map classifier $\mathcal{H}$ on a copy category $\mathbf{C}$ is *separable* when the copy transformation Δ belongs to $\mathcal{M}_{\mathcal{H}}$. The connection between copy categories and partial map classification is explored in depth by Cockett [13].

5.2 The category of circuits

In this section we review the circuit construction developed by Hensel and Spooner [18], and note the properties relevant to the subsequent sections. The construction differs from that of Katis, Sabadini and Walters [23] in two ways: the addition of initial states turns the 2-cell structure into a meaningful behavioral equivalence, and parameterizing the construction by a monad gives flexibility in the resulting notion of circuit. For a more detailed presentation, the reader is referred to [18].

Let $\mathbf{X}$ be a copy category, and (H, η, μ) a copy monad on $\mathbf{X}$:

Definition 5.2.1 *The bicategory of circuits in* $\mathbf{X}$ *is given by the following data:*

- *0-cells A are objects of* $\mathbf{X}$
- *1-cells $f : A \to B$ are pairs $(i : \top \to S,\ f : A \otimes S \to H(S \otimes B))$, where the identity id_A is the pair $(id : \top \to \top,\ c : A \otimes \top \to \top \otimes A)$ and the composition of $f : A \to B$ and $g : B \to C$ has initial state $\langle i_f, i_g \rangle : \top \to S_f \otimes S_g$ and state transformation*

$$\begin{tikzcd}
A\otimes(S_f\otimes S_g) \arrow[rr, dashed] \arrow[d, "a^{-1}"'] & & H(S_f\otimes S_g\otimes C) \\
(A\otimes S_f)\otimes S_g \arrow[d, "f\otimes 1"'] & & H^2(S_f\otimes S_g\otimes C) \arrow[u, "\mu"'] \\
H(S_f\otimes B)\otimes S_g \arrow[d, "\tau^L"'] & & H^2(S_f\otimes(S_g\otimes C)) \arrow[u, "H^2a^{-1}"'] \\
H(S_f\otimes B\otimes S_g) \arrow[r, "Ha"'] & H(S_f\otimes(B\otimes S_g)) \arrow[r, "H(1\otimes g)"'] & H(S_f\otimes H(S_g\otimes C)) \arrow[u, "H\tau^R"']
\end{tikzcd}$$

- *2-cells $f \Rightarrow g$ are morphisms $h : S_f \to S_g$ of for which*

$$\begin{tikzcd}
 & \top \arrow[dl, "f"'] \arrow[dr, "g"] & \\
S_f \arrow[rr, "h"'] & & S_g
\end{tikzcd}
\qquad
\begin{tikzcd}
A\otimes S_f \arrow[r, "1\otimes h"] \arrow[d, "f"'] & A\otimes S_g \arrow[d, "g"] \\
H(S_f\otimes B) \arrow[r, "H(h\otimes 1)"'] & H(S_f\otimes B)
\end{tikzcd}$$

with identities and composition in $\mathbf{X}$*.*

Taking $\mathbf{X}$ to be **Set** and H to be the monad $\_+1$ of exceptions yield circuits which are *partial* in that certain inputs (depending on the circuit state) may cause the circuit to halt without producing an output. In this setting a composite circuit halts when either of its components halt.

The presence of initial states has the effect that 2-cells are bisimulations, and bisimulation in each hom-category is just the equivalence relation induced by the 2-cells. In this way one obtains the category of partial circuits: [1]

Definition 5.2.2 $\mathrm{Circ}(\mathbf{X}, H)$ *is the category obtained by quotienting the bicategory of circuits by all 2-cells.*

The tensor in $\mathbf{X}$ induces a tensor in the circuit category which corresponds to the parallel combination of circuits. Given circuits $f : A \to B$ and $g : C \to D$, the circuit $f \otimes g$ is as follows:

$$\begin{tikzcd}
(A\otimes C)\otimes(S_f\otimes S_g) \arrow[rr, dashed] \arrow[d, "ex"'] & & H(S_f\otimes S_g\otimes(B\otimes D)) \\
(A\otimes S_f)\otimes(C\otimes S_g) \arrow[r, "f\otimes g"'] & H(S_f\otimes B)\otimes H(S_g\otimes D) \arrow[r, "\tau"'] & H(S_f\otimes B\otimes(S_g\otimes D)) \arrow[u, "H(ex)"']
\end{tikzcd}$$

The natural transformations associated with the copy structure of $\mathbf{X}$ also lift to the circuit category. For example, the copy transformation becomes the following circuit:

$$\begin{tikzcd}
A\otimes T \arrow[r, dashed] \arrow[d, "c"'] & H(T\otimes(A\otimes A)) \\
T\otimes A \arrow[r, "1\otimes\Delta"'] & T\otimes(A\otimes A) \arrow[u, "\eta"']
\end{tikzcd}$$

The associativity, symmetry and unit elimination maps lift in the same way. The counits for each object (if they exist) also lift in this way, but naturality is not preserved (viz. when the tensor is a product in $\mathbf{X}$). Consequently,

Proposition 5.2.3 *If* $\mathbf{X}$ *is a strict copy category and* H *is a copy monad, then* $\mathrm{Circ}(\mathbf{X}, H)$ *is a strict copy category.*

[1] If one formulates the bicategory of circuits without initial states, each hom-category has an initial object — the circuit with an empty state space — and so the category obtained by quotienting is trivial.

Consider now the copy propositions $x : A \to \top$ in $\mathrm{Circ}(\mathbf{X}, H)$. These may be seen as languages — i.e. $\mathcal{L}_x$ is the prefix-closed set of finite and infinite strings over A for which the circuit x does not halt. The induced idempotent $e_x : A \to A$ is the circuit which allows the strings of $\mathcal{L}_x$ to pass through, but halts on strings outside of $\mathcal{L}_x$. For a circuit $f : A \to B$ to satisfy the equation $f = e_x; f; e_y$ means that f aborts on sequences outside $\mathcal{L}_x$ and does not produce as output strings outside $\mathcal{L}_y$ — i.e. the behavior of f is within the bounds of the specifications x and y. Thus we refer to the category of proposition idempotents as the category of safe circuits:

$$\mathrm{SafeCirc}(\mathbf{X}, H) \stackrel{def}{=} \mathrm{PSplit}(\mathrm{Circ}(\mathbf{X}, H))$$

Consider further the strict maps of this category: those $f : x \to y$ which satisfy the equation $f; y = x$ in $\mathrm{Circ}(\mathbf{X}, H)$. Such circuits cannot halt when provided strings in $\mathcal{L}_x$, and furthermore translate those strings to strings in $\mathcal{L}_y$. As this property is a form of liveness, we refer to the resulting category of copy propositions as the category of safe and live circuits:

$$\mathrm{LiveCirc}(\mathbf{X}, H) \stackrel{def}{=} \mathrm{CProp}(\mathrm{Circ}(\mathbf{X}, H))$$

5.3 Embedding circuits in SProc

This section demonstrates the equivalence of the categories of safe and live circuits and of functional processes. We begin by embedding the category of simply-typed circuits into the category of deterministic processes. Through splitting propositional idempotents, this embedding lifts to an equivalence of the categories of safe circuits and of deterministic processes. Finally, the strict morphisms in this setting are shown to be the category of functional processes.

There is an obvious translation of partial circuits into synchronous processes with trivial types, as we will describe shortly. For this functor to be full and faithful, however, requires a tight correspondance between the monad which specifies partiality of circuits and the class of monics which specify the permission sets of the transition systems underlying the process construction.

Let H be a separable partial map classifier on $\mathbf{X}$ with the property that the category of classified arrows has pullbacks:

Definition 5.3.1 *Define* $\mathrm{Tran}(\mathbf{X}, H)$ *to be the full subcategory of* $\mathrm{Tran}(\mathbf{X})$ *whose objects* A *have their permission component* m_A *in* $\mathcal{M}_H$*.*

We will embed the category of H-partial circuits into the category of deterministic processes constructed upon H-classified transition systems:

Definition 5.3.2 *Let* $\mathrm{DSProc}(\mathbf{X}, H)$ *be the process category upon* $\mathrm{Tran}(\mathbf{X}, H)$ *with left legs given by the local monics* $\exists_H^{\mathcal{M}}$ *and equivalence given by the local isomorphisms* $\exists^{\mathcal{I}}$.

Given a circuit $f : A \to B$ of $\mathrm{Circ}(\mathbf{X}, H)$, we can take the associated partial map (m, α) in $\mathbf{X}$

$$\begin{tikzcd}
P \arrow[r, "\alpha"] \arrow[d, "m"'] \arrow[dr, phantom, "\lrcorner", very near start] & S\times B \arrow[d, "\eta"] \\
A\times S \arrow[r, "f"'] & H(S\times B)
\end{tikzcd}$$

and form the apex of a span in $\mathrm{Tran}(\mathbf{X}, H)$ with endpoints the "chaotic" transition systems upon A and B respectively:

$$\begin{tikzcd}
1 & & S \arrow[ll, "!"'] \arrow[rr, "!"] & & 1 \\
A\times 1 \arrow[u, "!"] \arrow[dddd, equal] & A\times S \arrow[l, "1\times !"'] & P \arrow[l, "m"'] \arrow[u, "\alpha;\pi"'] \arrow[r, "\alpha;!\times 1"] \arrow[d, "\langle m,\alpha\rangle"] & 1\times B \arrow[r, "c"] & B\times 1 \arrow[u, "!"'] \arrow[dddd, equal] \\
& & A\times S\times(S\times B) \arrow[ul, "\pi"'] \arrow[ur, "!\times\pi'"'] \arrow[dd, "1\times\pi'"] & & \\
& & & S\times B \arrow[uu, "!\times 1"'] \arrow[dd, "c"] & \\
& & A\times S\times B \arrow[ur, "\pi'\times 1"'] \arrow[dd, "sw"] & & \\
& & & B\times S \arrow[dr, "1\times !"] & \\
A\times 1 & & A\times B\times S \arrow[ll, "\pi\times !"] \arrow[uuuuul, "\pi\times 1"] \arrow[ur, "\pi'\times 1"] \arrow[rr, "\pi'\times !"'] & & B\times 1
\end{tikzcd}$$

We will use $F_0(A) \overset{F_L(f)}{\leftarrow} F_1(f) \overset{F_R(f)}{\rightarrow} F_0(A)$ to name the components of the above span, and F as the translation of circuits to spans.

Proposition 5.3.3 $F : \mathrm{Circ}(\mathbf{X}, H) \to \mathrm{DSProc}(\mathbf{X}, H)$ *is a functor.*

Proof. The diagram above shows that $(F_L(f) F_R(f))$ forms a span in $\mathrm{Tran}(\mathbf{X})$. To represent a process of $\mathrm{DSProc}(\mathbf{X}, H)$ we must have $m_{F_1(f)}$ in $\mathcal{M}_H$ and $F_L(f)$ in $\exists^{\mathcal{M}}$. First note that for any x, $\langle 1, x\rangle$ is in $\mathcal{M}_H$ since

$$\begin{tikzcd}
A \arrow[r, "\langle 1,h\rangle"] \arrow[d, "h"'] \arrow[dr, phantom, "\lrcorner", very near start] & A\times B \arrow[d, "h\times 1"] \\
B \arrow[r, "\Delta"'] & B\times B
\end{tikzcd}$$

is a pullback and Δ is in $\mathcal{M}_H$. Thus $\langle m, \alpha; \pi' \rangle; sw = \langle 1, \alpha; \pi' \rangle; m{\times}1; sw$ is in $\mathcal{M}_H$. Then $F_L(f)$ is seen to belong to $\exists^{\mathcal{M}}$ as follows:

$$
\begin{tikzcd}
A\times 1 \arrow[dd, "="] & A\times S \arrow[l, "1\times !"'] \arrow[dd, "="] & & P \arrow[ll, "m"'] \arrow[d, "{\langle m,\alpha;\pi' \rangle}"] \arrow[ddl, "m"'] \\
 & & & A\times S\times B \arrow[d, "sw"] \\
A\times 1 \arrow[d, "\pi'"] & A\times S \arrow[l, "1\times !"'] & A\times S \arrow[l, "="'] \arrow[dr, "\pi'"'] & A\times B\times S \arrow[d, "\pi'"] \\
1 & & & S \arrow[lll, "!"]
\end{tikzcd}
$$

To show that F preserves equality, we suppose $h : f \to g : A \to B$ is a 2-cell of Circ and show that it induces an $\exists^{\mathcal{I}}$ 2-cell $F(f) \to F(g)$ of DSProc. Forming the partial maps of f and g induces $h_P : P_f \to P_g$ in $\mathbf{X}$ such that

$$
\begin{tikzcd}
P_f \arrow[rr, "m_f"] \arrow[dr, dashed, "h_P"] \arrow[dd, "\alpha_f"'] & & A\times S_f \arrow[dr, "1\times h_S"] \arrow[dd, "f"] & \\
 & P_g \arrow[rr, "m_g"] \arrow[dd, "\alpha_g"] & & A\times S_g \arrow[dd, "g"] \\
S_f\times B \arrow[rr, "\eta"] \arrow[dr, "h_S\times 1"'] & & H(S_f\times B) \arrow[dr] & \\
 & S_g\times B \arrow[rr, "\eta"] & & H(S_g\times B)
\end{tikzcd}
$$

commutes, making h a morphism $F(f) \to F(g)$ in Tran. As η is cartesian the top face above is a pullback, and thus h is seen to belong to $\exists^{\mathcal{I}}$ as follows:

$$
\begin{tikzcd}
 & A\times S_f\times B \arrow[rr, "sw"] \arrow[dr, "\pi"] & & A\times B\times S_f \arrow[dr, "\pi'"] & \\
P_f \arrow[ur, "{\langle m,\alpha;\pi' \rangle}"] \arrow[rr, "m_f"] \arrow[d, "h_P"'] & & A\times S_f \arrow[rr, "\pi'"] \arrow[d, "1\times h_S"'] & & S_f \arrow[d, "h_S"] \\
P_g \arrow[rr, "m_g"] & & A\times S_g \arrow[rr, "\pi'"] & & S_g
\end{tikzcd}
$$

That $h : F(f) \to F(g)$ is a 2-cell in DSProc is verified componentwise by $1; \pi = \pi$ and $1; \pi' = \pi'$ for the labels, and $h_S; !_{S_g} = !_{S_f}$ for the states.

To show that F preserves identities, we suppose A in Circ and produce an $\exists^{\mathcal{I}}$ 2-cell $1_{F(A)} \to F(1_A)$ of DSProc. Recall that 1_A in Circ is given by $c; \eta : A{\times}1 \to H(1{\times}A)$. The morphism

$F_0(A) \to F_1(1_A)$ in Tran is then given as follows:

$$
\begin{tikzcd}
1 \arrow[dd, "="] & A\times 1 \arrow[l, "!"'] \arrow[rrr, "="] \arrow[dd, "="] & & & A\times 1 \arrow[dl, "\Delta\times\Delta"'] \arrow[dd, "\Delta\times 1"] \\
& & A\times 1\times(A\times 1) \arrow[r, "ex"] \arrow[d, "1\times c"] \arrow[dr, "1\times\pi"] & A\times A\times(1\times 1) \arrow[dr, "1\times\pi"] & \\
1 & A\times 1 \arrow[l, "c;\pi"] \arrow[ur, "\Delta"] \arrow[r, "\langle 1,c\rangle"'] & A\times 1\times(1\times A) \arrow[r, "1\times\pi'"'] & A\times 1\times A \arrow[r, "sw"] & A\times A\times 1
\end{tikzcd}
$$

It belongs to $\exists^{\mathcal{I}}$ as the permission and state components are isomorphisms, and is easily checked to be a 2-cell.

To show that F preserves composition, we suppose $f : A \to B$ and $g : B \to C$ in Circ and produce an $\exists^{\mathcal{I}}$ 2-cell $h : F(f); F(g) \to F(f; g)$.

$$
\begin{tikzcd}
& & F_1(f;g) \arrow[dddll] \arrow[dddrr] & & \\
& & Z \arrow[u, dashed, "h"] \arrow[dl, "x"'] \arrow[dr, "y"] & & \\
& F_1(f) \arrow[dl] \arrow[dr] & & F_1(g) \arrow[dl] \arrow[dr] & \\
F_0(A) & & F_0(B) & & F_0(C)
\end{tikzcd}
$$

First we construct the partial map corresponding to $f;g$:

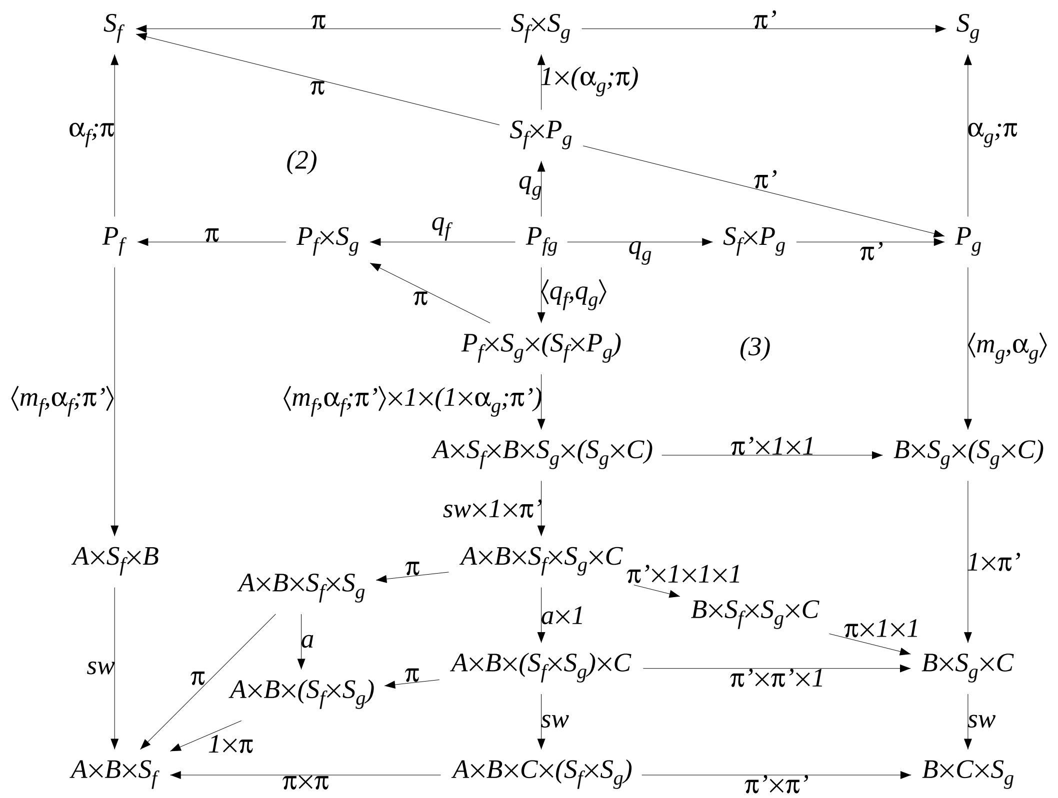
$$
\begin{tikzcd}
P_{fg} \arrow[r, "q_f"] \arrow[dddd, "q_g"'] \arrow[dr, phantom, very near start, "\lrcorner"] & P_f\times S_g \arrow[r, "m_f\times 1"] \arrow[d, "="] & A\times S_f\times S_g \arrow[r, "a"] & A\times(S_f\times S_g) \arrow[d, "a^{-1}"] \\
& P_f\times S_g \arrow[rr, "m_f\times 1"] \arrow[d, "\alpha_f\times 1"'] & & A\times S_f\times S_g \arrow[d, "f\times 1"] \\
(1) & S_f\times B\times S_g \arrow[rr, "\eta\times 1"] \arrow[d, "="] & & H(S_f\times B)\times S_g \arrow[d, "\tau^L"] \\
& S_f\times B\times S_g \arrow[rr, "\eta"] \arrow[d, "a"] & & H(S_f\times B\times S_g) \arrow[d, "Ha"] \\
S_f\times P_g \arrow[r, "1\times m_g"] \arrow[d, "a\times\alpha_g"'] & S_f\times(B\times S_g) \arrow[d, "1\times g"] & & H(S_f\times(B\times S_g)) \arrow[d, "H(1\times g)"] \\
S_f\times(S_g\times C) \arrow[r, "1\times\eta"] \arrow[d, "="] & S_f\times H(S_g\times C) \arrow[d, "\tau^R"] & & H(S_f\times H(S_g\times C)) \arrow[d, "H\tau^R"] \\
S_f\times(S_g\times C) \arrow[r, "\eta"] \arrow[d, "a^{-1}"'] & H(S_f\times(S_g\times C)) \arrow[d, "Ha^{-1}"'] & & H^2(S_f\times(S_g\times C)) \arrow[d, "H^2a^{-1}"] \\
S_f\times S_g\times C \arrow[r, "\eta"] \arrow[d, "="] & H(S_f\times S_g\times C) \arrow[rr, "\eta"] & & H^2(S_f\times S_g\times C) \arrow[d, "\mu"] \\
S_f\times S_g\times C \arrow[rrr, "\eta"'] & & & H(S_f\times S_g\times C)
\end{tikzcd}
$$

Now construct the pullback (Z,x,y) componentwise: the state component is $(S_f\times S_g,\pi,\pi')$; the label component is $(A\times B\times C,\pi,\pi'\times 1)$; and the permission component appears below.

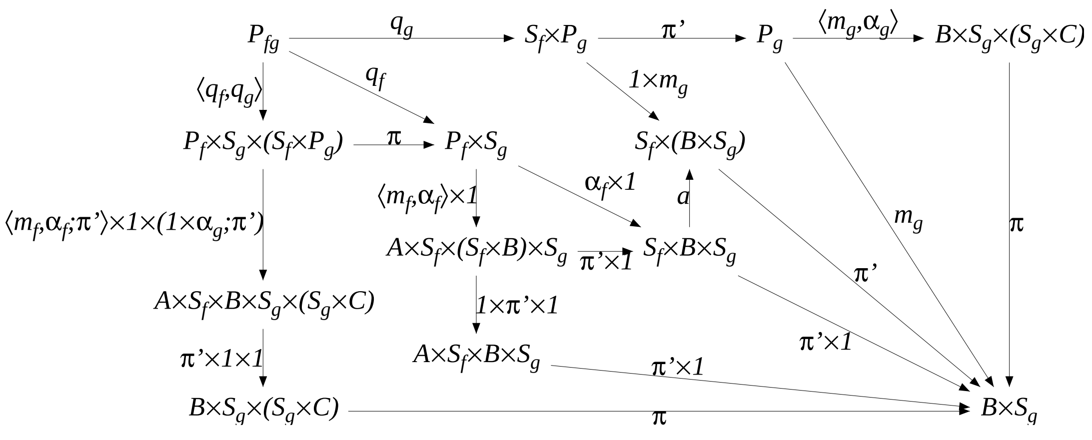
$$
\begin{tikzcd}
P_{fg} \arrow[r, "q_f"] \arrow[ddd, "q_g"'] \arrow[dr, phantom, very near start, "\lrcorner"] & P_f\times S_g \arrow[rr, "="] \arrow[d, "\alpha_f\times 1"'] & & P_f\times S_g \arrow[rr, "\pi"] \arrow[d, "\alpha_f\times 1"'] & & P_f \arrow[d, "\alpha_f"] \\
& S_f\times B\times S_g \arrow[rr, "="] \arrow[dd, "a"] & & S_f\times B\times S_g \arrow[rr, "\pi"] \arrow[d, "sw"'] & & S_f\times B \arrow[d, "="] \\
& & & S_f\times S_g\times B \arrow[rr, "\pi\times 1"] \arrow[dl, "c"'] \arrow[dr, "\pi'\times 1"] & & S_f\times B \arrow[d, "!\times 1"] \\
S_f\times P_g \arrow[r, "1\times m_g"] \arrow[d, "\pi'"'] & S_f\times(B\times S_g) \arrow[r, "\sim"] \arrow[d, "\pi'"'] & B\times(S_f\times S_g) \arrow[dr, "1\times\pi'"'] & & S_g\times B \arrow[r, "!\times 1"] \arrow[dl, "c"] & 1\times B \arrow[d, "c"] \\
P_g \arrow[r, "m_g"'] & B\times S_g \arrow[rr, "="] & & B\times S_g \arrow[rr, "1\times !"'] & & B\times 1
\end{tikzcd}
$$

The transition system Z is given by the unique morphisms m_Z and α_Z such that $m_Z;x_\Sigma\times x_S =$

$x_P; m_{Ff}$ and $m_z; y_\Sigma \times y_S = y_P; m_{Fg}$, and $\alpha_Z; x_S = x_P; \alpha_{Ff}$ and $\alpha_Z; y_S = y_P; \alpha_{Fg}$:



Note that (2) is (1); π. As a map into a product, we verify (3) componentwise: i.e. (3); π



and (3); π' commute:

$$\begin{tikzcd}
P_{fg} \arrow[r, "q_g"] \arrow[d, "\langle q_f, q_g\rangle"'] & S_f\times P_g \arrow[r, "\pi'"] \arrow[ddr, "1\times\alpha_g"] & P_g \arrow[r, "\langle m_g,\alpha_g\rangle"] \arrow[ddddr, "\alpha_g"] & B\times S_g\times(S_g\times C) \arrow[dddd, "\pi'"] \\
P_f\times S_g\times(S_f\times P_g) \arrow[ur, "\pi'"] \arrow[d, "\langle m_f,\alpha_f;\pi'\rangle\times 1\times(1\times\alpha_g;\pi')"'] & & & \\
A\times S_f\times B\times S_g\times(S_g\times C) \arrow[dd, "\pi'\times 1\times 1"'] & & S_f\times(S_g\times C) \arrow[ddr, "\pi'"] & \\
 & & & \\
B\times S_g\times(S_g\times C) \arrow[rrr, "\pi'"'] & & & S_g\times C
\end{tikzcd}$$

The map $h : Z \to F(f;g)$ is then given as follows:

$$\begin{tikzcd}
S_f\times S_g \arrow[rr, "="] & & S_f\times S_g \\
S_f\times(S_g\times C) \arrow[u, "1\times\pi"] \arrow[rr, "="] & & S_f\times(S_g\times C) \arrow[u, "a^{-1};\pi"'] \\
P_{fg} \arrow[u, "q_g;1\times\alpha_g"] \arrow[rr, "="] & & P_{fg} \arrow[u, "q_g;1\times\alpha_g"'] \\
P_f\times S_g\times(S_f\times P_g) \arrow[u, leftarrow, "\langle q_f,q_g\rangle"] \arrow[rr, "="] & & P_f\times S_g\times(S_f\times P_g) \arrow[u, leftarrow, "\langle q_f,q_g\rangle"'] \\
A\times S_f\times(S_f\times B)\times S_g\times(S_f\times(S_g\times C)) \arrow[u, leftarrow, "\langle m_f,\alpha_f\rangle\times 1\times(1\times\alpha_g)"] \arrow[rr, "\pi\times 1\times 1"] & & A\times S_f\times S_g\times(S_f\times(S_g\times C)) \arrow[u, leftarrow, "m_f\times 1\times(1\times\alpha_g)"'] \arrow[dl, "1\times\pi'"'] \arrow[d, "a\times a^{-1}"'] \\
A\times S_f\times B\times S_g\times(S_g\times C) \arrow[u, leftarrow, "1\times\pi'\times 1\times\pi'"] \arrow[r, "\pi\times 1\times 1"] & A\times S_f\times S_g\times(S_g\times C) \arrow[d, "1\times\pi'"] & A\times(S_f\times S_g)\times(S_f\times S_g\times C) \arrow[dd, "1\times\pi'"'] \\
A\times B\times S_f\times S_g\times C \arrow[u, leftarrow, "sw\times 1\times\pi'"] \arrow[r, "\pi\times 1\times 1\times 1"] & A\times S_f\times S_g\times C \arrow[dr, "a\times 1"] & \\
A\times B\times(S_f\times S_g)\times C \arrow[u, leftarrow, "a\times 1"] \arrow[rr, "\pi\times 1\times 1"] \arrow[d, "sw"] & & A\times(S_f\times S_g)\times C \arrow[d, "sw"'] \\
A\times B\times C\times(S_f\times S_g) \arrow[rr, "\pi\times 1\times 1"] & & A\times C\times(S_f\times S_g)
\end{tikzcd}$$

It is an $\exists^{\mathcal{I}}$-morphism as the state and permission components are isomorphisms, and is easily seen to be a 2-cell $F(f);F(g) \to F(f;g)$. $\square$

The functor F encoding circuits as processes also preserves the copy structure of circuits, and thus lies in the domain of the 2-functor PSplit.

Proposition 5.3.4 $F : \mathrm{Circ}(\mathbf{X}, H) \to \mathrm{DSProc}(\mathbf{X}, H)$ *is a copy functor.*

Proof. F preserves the tensor structure as $F_0(1) = 1$ and there is a natural isomorphism $F(A \otimes B) \to FA \otimes FB$ in Tran corresponding to the exchange map. To see the latter suppose $f : A \to B$ and $g : C \to D$ in Circ, and construct the partial map of $f \otimes g$ as follows:

$$
\begin{tikzcd}
P_f \times P_g \arrow[r, "m_f \times m_g"] \arrow[d, "="'] & A \times S_f \times (C \times S_g) \arrow[r, "ex"] & A \times C \times (S_f \times S_g) \arrow[d, "ex"] \\
P_f \times P_g \arrow[rr, "m_f \times m_g"] \arrow[d, "\alpha_f \times \alpha_g"'] & & A \times S_f \times (C \times S_g) \arrow[d, "f \times g"] \\
S_f \times B \times (S_g \times D) \arrow[rr, "\eta \times \eta"] \arrow[d, "="'] & & H(S_f \times C) \times H(S_g \times D) \arrow[d, "\tau"] \\
S_f \times B \times (S_g \times D) \arrow[rr, "\eta"] \arrow[d, "ex"'] & & H(S_f \times B \times (S_g \times D)) \arrow[d, "Hex"] \\
S_f \times S_g \times (B \times D) \arrow[rr, "\eta"'] & & H(S_f \times S_g \times (B \times D))
\end{tikzcd}
$$

The required isomorphism in Tran is then:

$$
\begin{tikzcd}
S_f \times S_g \arrow[r, "="] & S_f \times S_g \\
S_f \times S_g \times (B \times D) \arrow[u, "\pi"] & \\
S_f \times B \times (S_g \times D) \arrow[u, "ex"] \arrow[r, "="] & S_f \times B \times (S_g \times D) \arrow[uu, "\pi \times \pi"'] \\
P_f \times P_g \arrow[u, "\alpha_f \times \alpha_g"] \arrow[r, "="] \arrow[d, "\langle m_f \times m_g, \alpha_f \times \alpha_g \rangle"'] & P_f \times P_g \arrow[u, "\alpha_f \times \alpha_g"'] \arrow[d, "\langle m_f \times \alpha_f, m_g \times \alpha_g \rangle"] \\
A \times S_f \times (C \times S_g) \times (S_f \times B \times (S_g \times D)) \arrow[r, "ex"] \arrow[d, "ex \times ex"'] & A \times S_f \times (S_f \times B) \times (C \times S_g \times (S_g \times D)) \arrow[d, "1 \times \pi' \times (1 \times \pi')"] \\
A \times C \times (S_f \times S_g) \times (S_f \times S_g \times (B \times D)) \arrow[d, "1 \times \pi'"'] & A \times S_f \times B \times (C \times S_g \times D) \arrow[d, "sw \times sw"] \\
A \times C \times (S_f \times S_g) \times (B \times D) \arrow[d, "sw"'] & A \times B \times S_f \times (C \times D \times S_g) \arrow[d, "ex"] \\
A \times C \times (B \times D) \times (S_f \times S_g) \arrow[r, "ex \times 1"'] & A \times B \times (C \times D) \times (S_f \times S_g)
\end{tikzcd}
$$

$\square$

The fact that the encoding of circuits as processes is an embedding relies on the use of a separable partial map classifier.

Proposition 5.3.5 $F : \mathrm{Circ}(\mathbf{X}, H) \to \mathrm{DSProc}(\mathbf{X}, H)$ *is full and faithful.*

Proof. To show F is faithful, it is sufficient to show that for every $\exists^{\mathcal{I}}$ 2-cell $e : (h, k) \to F(f)$ of DSProc, there is a circuit f' and 2-cell $f' \to f$ with $F(f') = (h, k)$. Note that $e_\Sigma = \langle h_\Sigma, k_\Sigma \rangle : \Sigma_C \to A{\times}B$, and thus we have the following morphism of partial maps:

$$\begin{tikzcd}
 & & \Sigma_C\times S_C \arrow[dll, "\pi'"'] \arrow[dl, "h\times 1"] & & P_C \arrow[ll, "m_C"'] \arrow[dr, "\langle\alpha_C{,}m_C;\pi;k\rangle"] \arrow[dd, "e_P"] \arrow[ddr, phantom, "(1)"] & \\
S_C \arrow[dd, "e_S"'] & A\times S_C \arrow[l, "\pi'"'] \arrow[dd, "1\times e_S"] & & & & S_C\times B \arrow[dd, "e_S\times 1"] \\
 & & A\times B\times S_f \arrow[dll, "\pi'"'] \arrow[dl, "\pi\times 1"] & & P_f \arrow[ll, "\langle m_f{,}\alpha_f;\pi'\rangle;sw"'] \arrow[dlll, "m_f"] \arrow[dr, "\alpha_f"'] & \\
S_f & A\times S_f \arrow[l, "\pi'"] & & & & S_f\times B
\end{tikzcd}$$

Here we see that $(1); \pi$ commutes since e is a morphism of Tran, and $(1); \pi'$ commutes as follows:

$$\begin{tikzcd}
P_C \arrow[rr, "m_C"] \arrow[d, "e_P"'] & & \Sigma_C\times S_C \arrow[r, "\pi"] \arrow[d, "e_\Sigma\times e_S"] & \Sigma_C \arrow[d, "e"] \arrow[ddr, "k"] & \\
P_f \arrow[r, "\langle m_f{,}\alpha_f;\pi'\rangle"] \arrow[dr, "\alpha_f"'] & A\times S_f\times B \arrow[r, "sw"] \arrow[drrr, "\pi'"] & A\times B\times S_f \arrow[r, "\pi"] & A\times B \arrow[dr, "\pi'"'] & \\
 & S_f\times B \arrow[rrr, "\pi'"'] & & & B
\end{tikzcd}$$

The fact that $e \in \exists^{\mathcal{I}}$ implies that $m_c; h{\times}1$ is a pullback of m_f along e_P and thus an element of $\mathcal{M}$. Taking f' to be the associated circuit, we have $e_S : f' \to f$ in Circ. Finally, we have the following morphism $C \to F_1(f')$ of Tran

$$\begin{tikzcd}
\Sigma_C\times S_C \arrow[dd, "e_\Sigma\times 1"'] & & & P_C \arrow[lll, "m_C"'] \arrow[dl, "m_C"] \arrow[dd, "="] \arrow[r, "\alpha_C"] & S_C \arrow[dd, "="] \\
 & & \Sigma_C\times S_C \arrow[ull, "="'] \arrow[dl, "\langle h\times 1{,}\pi;k\rangle"'] & & \\
A\times B\times S_C & A\times S_C\times B \arrow[l, "sw"] & & P_C \arrow[ul, "m_C"'] \arrow[ll, "\langle m_C;h\times 1{,}\langle\alpha_C;m_C;\pi;k\rangle;\pi'\rangle"] \arrow[r, "\alpha_C"'] & S_C
\end{tikzcd}$$

which is easily seen to be an $\exists^{\mathcal{I}}$ 2-cell $F(f') \to (h, k)$.

To show F is full, we suppose $(f, g) : FA \to FB$ in DSProc and construct a circuit $h : A \to B$ and an $\exists^{\mathcal{I}}$ 2-cell $(f, g) \to F(h)$ in DSProc. First note that as $f \in \exists^{\mathcal{M}}$, (2) below is an $\exists^{\mathcal{M}}$-pullback and thus $m_C; f_\Sigma{\times}1$ belongs to $\mathcal{M}$.

$$\begin{tikzcd}
S_C \arrow[dd, "!"'] & & \Sigma_C{\times}S_C \arrow[ll, "\pi'"'] \arrow[dl, "f_\Sigma{\times}1"] & P_C \arrow[l, "m_C"'] \arrow[dd, dashed] \\
& A{\times}S_C \arrow[ul, "\pi'"] \arrow[d, "1{\times}!"] \arrow[rr, phantom, "(2)"] & & \\
1 & A{\times}1 \arrow[l, "\pi'"] & & A{\times}1 \arrow[ll, "="]
\end{tikzcd}$$

Take $h : A \to B$ to be the circuit associated with the partial map $(m_C; f_\Sigma{\times}1, l\alpha_C, m_C; \pi; g_\Sigma\rangle)$. The required $\exists^{\mathcal{I}}$ 2-cell of DSProc is then given by the following morphism $C \to F(h)$ of Tran:

$$\begin{tikzcd}
\Sigma_C{\times}S_C \arrow[d, "\langle f_\Sigma, g_\Sigma\rangle{\times}1"'] \arrow[dr, "\langle f_\Sigma{\times}1, \pi; g_\Sigma\rangle"] & & P_C \arrow[ll, "m_C"'] \arrow[r, "\alpha_C"] \arrow[d, "="] & S_C \arrow[d, "="] \\
A{\times}B{\times}S_C & A{\times}S_C{\times}B \arrow[l, "sw"] & P_C \arrow[l, "\langle m_C; f_\Sigma{\times}1, \langle \alpha_C, m_C; \pi; g_\Sigma\rangle; \pi'\rangle"] \arrow[r, "\langle \alpha_C, m_C; \pi; f_\Sigma\rangle; \pi"'] & S_C
\end{tikzcd}$$

□

We now show that the embedding of circuits into deterministic processes lifts via the idempotent completion to an equivalence between the categories of safe circuits and deterministic processes. The reason is that copy idempotents of $\mathrm{Circ}(\mathbf{X}, H)$ are taken by the functor to split idempotents in $\mathrm{DSProc}(\mathbf{X}, H)$.

We begin by noting a general class of idempotents in process categories, followed by a specific class of idempotents in SProc.

Lemma 5.3.6 *If $f : C \to A$ is in $\mathcal{C}_0 \cap \mathcal{C}_1$ and $1; f = 1; f$ is an $\mathcal{E}$-pullback, then $(f, f) : A \rightsquigarrow A$ is a split idempotent in* $\mathrm{Proc}(\mathbf{X}, \mathcal{E}, \mathcal{C}_0, \mathcal{C}_1)$.

Of course any monic f satisfies this property. Furthermore, if $\mathcal{E}$ contains all retractions then any f will work since the projections from the pullback of f and f serve as $\mathcal{E}$ 2-cells $(f, f); (f, f) \to (f, f)$. In the category of transition systems, the *local* monics also have this property.

Lemma 5.3.7 *In* $\mathrm{Tran}(\mathbf{X})$, *if $f \in \exists^{\mathcal{M}}$ then $1; f = 1; f$ is an $\exists^{\mathcal{I}}$-pullback.*

Proof. Forming the pullback of f along itself we have the following situation:

$$\begin{tikzcd}
& A \arrow[dl, "="'] \arrow[d, dashed, "h"] \arrow[dr, "="] & \\
A \arrow[dr, "f"'] & Q \arrow[l, "q"'] \arrow[r, "q'"] & A \arrow[dl, "f"] \\
& B &
\end{tikzcd}$$

The fact that $h; q = 1$ induces w in the diagram below which makes $(m_a; \pi'); h_S = w; y$ a pullback.

$$\begin{tikzcd}
P_A \arrow[rrr, "m_A;\pi'"] \arrow[dr, "="] \arrow[dd, "h_P"'] & & & S_A \arrow[dd, "h_S"] \\
& \cdot \arrow[urr] \arrow[dd, "w"] & & \\
P_Q \arrow[rrr, "m_Q;\pi'"] \arrow[dr, "z"] \arrow[dd, "q_P"'] & & & S_Q \arrow[dd, "q_S"] \\
& \cdot \arrow[urr, "y"'] \arrow[dl, "x"] & & \\
P_A \arrow[rrr, "m_A;\pi'"'] & & & S_A
\end{tikzcd}$$

Since q belongs to $\exists^{\mathcal{M}}$, z is monic and thus the top square is a pullback as required. $\square$

Consequently, any process (f, f) is a split idempotent in $\mathrm{DSProc}(\mathbf{X}, H)$.

Returning to the embedding of circuits, we note that the copy propositions of $\mathrm{Circ}(\mathbf{X}, H)$ correspond exactly to the objects of $\mathrm{Tran}(\mathbf{X}, H)$:

$$\begin{tikzcd}
P_x \arrow[r, "\alpha_x"] \arrow[d, "m_x"'] & S_x \arrow[d, "\eta"] \\
A\times S_x \arrow[r, "x"'] & HS_x
\end{tikzcd}$$

We define $G_0(x)$ to be the transition systems associated with $x : A \to 1$ of Circ, and $G_1(x) = (1_A, !_S) : G_0(x) \to F_0(A)$ the obvious morphism of Tran.

Lemma 5.3.8 *For $x : A \to 1$ of $\mathrm{Circ}(\mathbf{X}, H)$, $(G_1(x), G_1(x)) = F(e_x) : F(A) \to F(A)$ in $\mathrm{FSProc}(\mathbf{X}, H)$.*

Proof. We exhibit an $\exists^{\mathcal{I}}$ 2-cell $h : (G_1(x), G_1(x)) \to F(e_x)$. First construct the partial map of e_x:

$$\begin{tikzcd}
P \arrow[r, "\langle 1,m\rangle"] \arrow[d, "m"'] & P\times(A\times S) \arrow[r, "\alpha\times\pi"] \arrow[d, "m\times 1"] & S\times A \arrow[r, "="] \arrow[d, "\eta\times 1"] & S\times A \arrow[d, "\eta"] \\
A\times S \arrow[r, "\Delta"'] & A\times S\times(A\times S) \arrow[r, "x\times\pi"'] & GS\times A \arrow[r, "\tau_L"'] & G(S\times A)
\end{tikzcd}$$

The $\exists^{\mathcal{I}}$-morphism $h : G_0(x) \to F_1(e_x)$ of Tran is then as follows:

$$\begin{tikzcd}
A\times S \arrow[d, "\Delta\times 1"'] \arrow[dr, "\langle 1,\pi\rangle"] & & P \arrow[ll, "m"'] \arrow[r, "\alpha"] \arrow[d, "="] & & S \arrow[d, "="] \\
A\times A\times S & A\times S\times A \arrow[l, "sw"] & P \arrow[l, "\langle m,\langle\alpha,m;\pi\rangle;\pi'\rangle"] \arrow[r, "\langle\alpha,m;\pi\rangle"'] & S\times A \arrow[r, "\pi"'] & S
\end{tikzcd}$$

It is a 2-cell as $\Delta; \pi = 1_A = \Delta; \pi'$ and $1_S; !_S = !_S$. Note that $G_1(x)$ belongs to $\exists^{\mathcal{M}}$ as $F_L(e_x)$ is in $\exists^{\mathcal{M}}$ and h is in $\exists^{\mathcal{I}}$. $\square$

Thus PSplit(F) is a full and faithful functor SafeCirc$(\mathbf{X}, H) \to$ DSProc$(\mathbf{X}, H)$. Furthermore, every object A of DSProc$(\mathbf{X}, H)$ is isomorphic to PSplit$(F)(e_x)$, where $x : \Sigma_A \to 1$ is its classifying circuit, since the spans $(G_1(x), 1_A) : F(\Sigma_A) \to A$ and $(1_A, G_1(x)) : A \to F(\Sigma_A)$ are a splitting of $F(e_x)$. Thus:

Proposition 5.3.9 PSplit(F) : SafeCirc$(\mathbf{X}, H) \to$ DSProc$(\mathbf{X}, H)$ *is an equivalence of categories.*

As DSProc is a strict copy category, we can ask what are the strict morphisms. First we note that the counit of an object A is given by the process $(1_A, !_A) : A \to 1$.

Define FSProc$(\mathbf{X}, H) \stackrel{def}{=}$ Proc(Tran$(\mathbf{X}, H), \exists^{\mathcal{I}}, \exists^{\mathcal{I}})$ to be the subcategory of DSProc$(\mathbf{X}, H)$ whose processes have their left leg a local isomorphism.

Proposition 5.3.10 *The strict morphisms of* DSProc$(\mathbf{X}, H)$ *are given by* FSProc$(\mathbf{X}, H)$.

Proof. If e is in $\exists^{\mathcal{I}}$ then any process $(e, f) : A \to B$ is strict, as 1 is final in Tran$(\mathbf{X}, H)$:

$$\begin{tikzcd}
& A \arrow[ld, "="'] \arrow[rr, "!"] & & 1 \\
A & & B \arrow[ru, "!"'] & \\
& \cdot \arrow[lu, "e"] \arrow[uu, "e"] \arrow[ru, "f"'] & &
\end{tikzcd}$$

Conversely, if $(f, g) : A \to B$ is strict in FSProc then we have the following commuting diagram with e and e' in $\exists^{\mathcal{I}}$:

$$\begin{tikzcd}
& A \arrow[ld, "="'] \arrow[rrd, "!"] & & \\
A & \cdot \arrow[u, "e"] \arrow[d, "e'"] & & 1 \\
& \cdot \arrow[lu, "f"] \arrow[r, "g"'] & B \arrow[ru, "!"'] &
\end{tikzcd}$$

Thus (f, g) in the form $(e, e'; g) : A \to B$ belongs to FSProc$(\mathbf{X}, H)$. □

5.4 Summary

We have shown that a category of partial circuits embeds into the subcategory of SProc whose objects are simply alphabets. One obtains a category of safe circuits by splitting the propositional idempotents: this category is (up to equivalence) the category of deterministic processes of section 2.5.1. We thus have the following statement:

Theorem 5.4.1 *The following are equivalent:*

· SafeCirc(**Set**), *the category of safe circuits;*

· DSProc(**Set**), *the category of deterministic synchronous processes;*

· Par(Tree(**Set**)), *the category of partial maps upon trees.*

Within the category of safe circuits, one can equationally isolate those which satisfy a certain liveness property. This category of live circuits is the category of functional processes of section 2.5.1. In summary,

Theorem 5.4.2 *The following are equivalent:*

· FSProc(**Set**), *the category of functional synchronous processes;*

· LiveCirc(**Set**), *the category of live circuits;*

· Tree(**Set**), *the category of trees;*

These results provide a basis for programming processes in SProc: One begins by writing simply-typed circuits as elements of a coalgebraic datatype. More refined safety types are then introduced as certain (idempotent) circuits. Finally, justifying the assignment of safety and liveness types amounts to verifying equations involving the circuit and its (proposed) type.

Chapter 6

Conclusion

We have presented a construction of process categories as categories of generalized relations: one begins with a category of process models and a cover system expressing process equivalence; the category of processes is then obtained by a span construction quotiented by the cover system. We have shown that the construction is a 2-functor and (since the structure in a process category arises from related structure in it's model category) the task of establishing functorial properties of upon process categories is largely the task of demonstrating that the underlying structure lies in the domain of the 2-functor

We have shown that Abramsky's category SProc of synchronous processes arises as a process category from each of the standard models of interleaved concurrency — from transition systems to trees — and that SProc modulo trace equivalence is simply the category of relations upon trees. We have also shown how various categories of games and strategies arise through our construction.

We have presented an approach to adding asynchrony in the construction of process categories: specifically, one uses as a model category the Kleisli category of a suitable delay monad. Kleisli categories are not generally suitable as model categories, and the isolation of those monads which support the process construction represents a significant technical development. We use these techniques to provide a reconstruction of Abramsky's category ASProc of asynchronous which resembles Milner's approach to building asynchrony on top of synchrony in process calculus [26]. As a consequence, we have shown that cover systems for weak bisimulation and weak trace equivalence are given by a distributive law for the delay monad.

We have isolated a subcategory of SProc which can be characterized algebraicly as a category of circuits satisfying safety and liveness properties. This is the category of maps of SProc modulo trace equivalence, or equivalently the category of trees.

6.1 Future research

- complete construction of game theory category
- games upon trees
- alternatives for asynchrony
- game circuits

6.1.1 Generalizing the process construction

Although not warranted by our examples, there is an obvious generalization to the process construction: the model category need only have pullbacks involving the morphisms used as span legs. Thus we could replace the cover system parameters to the process construction with a span system:

Definition 6.1.1 *A span system on a category* $\mathbf{X}$ *is a pair* $(\mathcal{L}_0, \mathcal{L}_1)$ *of classes of morphisms of* $\mathbf{X}$ *such that* $\mathcal{L}_0$ *and* $\mathcal{L}_1$ *contain all isomorphisms and are closed under composition, and given any pair* (f, g) *of morphisms in* $\mathcal{L}_0 \times \mathcal{L}_1$ *the pullback*

$$\begin{tikzcd} \cdot \arrow[r, "g_1"] \arrow[d, "f_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & \cdot \arrow[d, "f"] \\ \cdot \arrow[r, "g"'] & \cdot \end{tikzcd}$$

exists with (f_1, g_1) *in* $\mathcal{L}_0 \times \mathcal{L}_1$*.*

One could further weaken the requirements on the cover system for equivalence by requiring that it is closed to pullback along the morphisms of the span system. Providing such flexibility may make it easier to define suitable delay monads for game-like models categories.

6.2 Specification structures

A specification structure is a faithful functor into a process category... The idea is that one has a chain of faithful functors: at the bottom sits a compact-closed process category with a rudimentary type structure, and each successive level represents a refinement of the type structure ... placing stronger demands on a process being included...

Although we do not provide an explicit treatment of specification structures in our setting, we believe one can acheive the same effect by adding structure to the model category and then using span systems to impose discipline...

Bibliography

[1] Samson Abramsky. Interaction categories (extended abstract). In *Theory and Formal Methods Workshop*. Springer Verlag, 1993.

[2] Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. Interaction categories and the foundation of typed concurrent programming. In M.Broy, editor, *Deductive Program Design: Proceedings of the 1994 Marktoberdorf Summer School*, NATO ASI Series F: Computer and System Sciences. Springer Verlag, 1994.

[3] Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543–574, 1994.

[4] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Games and full abstraction for pcf. *Lecture Notes in Computer Science*, 789:1–15, 1994. Proceedings of TACS '94.

[5] Michael Barr. *-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1:159–178, 1991.

[6] Michael Barr and Charles Wells. *Toposes, Triples and Theories*, volume 278 of *Grundlehren der mathematischen Wissenschaften*. Springer Verlag, 1985.

[7] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall, 1990.

[8] J. Bénabou. Introduction to bicategories. *Lecture Notes in Mathematics*, 40:1–77, 1967.

[9] Richard Blute, Prakash Panangaden, and R. A. G. Seely. Fock space: A model of linear exponential types. Manuscript, revised version of MFPS IX paper, 1994.

[10] Aurelio Carboni. Bicategories of partial maps. *Cahier Topologie Geom Differentielle Categoriques*, 28(2):111–126, 1897.

[11] J.R.B. Cockett. List-arithmetic open categories: Locoi. *Journal of Pure and Applied Algebra*, 66:1–29, 1990.

[12] J.R.B. Cockett. Introduction to distributive categories. *Mathematical Structures in Computer Science*, 3:277–307, 1993.

[13] J.R.B. Cockett. Copy categories. Submitted for publication, 1995.

[14] J.R.B. Cockett and R.A.G. Seely. Weakly distributive categories. *London Mathematical Society Lecture Notes*, 177:45–65, 1992.

[15] J.R.B. Cockett and David Spooner. Sproc categorically. *Lecture Notes in Computer Science*, 836:146–159, 1994. Proceedings of Concur '94.

[16] J.R.B. Cockett and David Spooner. Categories for synchrony and asynchrony. *Electronic Notes in Theoretical Computer Science*, 1:495–520, 1995.

[17] J.R.B. Cockett and David Spooner. Constructing process categories. *Theoretical Computer Science*, 1997. To appear.

[18] Ulrich Hensel and David A. Spooner. A view on implementing processes: categories of circuits. In *ADT '95*, Lecture Notes in Computer Science. Springer Verlag, 1996.

[19] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[20] Andre Joyal and I. Moerdijk. *Algebraic Set Theory*, volume 220 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1995.

[21] Andre Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation and open maps. In *Logic in Computer Science*. IEEE, 1993.

[22] P. Katis. *Categories and bicategories of processes*. PhD thesis, The University of Sydney, 1996.

[23] P. Katis, N. Sabadini, and R.F.C Walters. The bicategory of circuits. In C. Barry Jay, editor, *The Australian Theory Seminar*, pages 89–108, Sydney, 1994. University of Technology.

[24] G.M. Kelly. On clubs and data-type constructors. In *Applications of Categories in Computer Science*. Cambridge University Press, 1992.

[25] H. Lindner. A remark on mackey-functors. *Manuscripta Mathematica*, 18:273–278, 1976.

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

[27] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[28] Robert Paré. Simply connected limits. *Canadian Journal of Mathematics*, XLII(4):731–746, 1990.

[29] Duško Pavlović. Convenient categories of processes and simulations 1: modulo strong bisimilarity. Available by ftp from `beauty.doc.ic.ac.uk` as `papers/Pavlovic/CCPS1.ps.gz`, January 1995.

[30] V.R. Pratt. Chu spaces and their interpretation as concurrent objects. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 392–405. Springer-Verlag, 1995.

[31] E. Robinson and G. Rosolini. Categories of partial maps. *Information and Computation*, ?(2):95–130, 1988.

[32] G. Rosolini. Representation theorems for p-categories. *Lecture Notes in Mathematics*, 1348:307–315, 1988. Categorical algebra and it's applications (Louvain-La-Neuve, 1897).

[33] Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. A classification of models for concurrency. *Lecture Notes in Computer Science*, 715:82–96, 1993. Proceedings of Concur '93.

[34] Charles Wells. Sketches: Outline with references. Available by ftp from `ftp.cwru.edu` as `math/wells/sketch.ps`, February 1994.