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

Abstract

Categories of interacting processes have become important in the semantics of concurrency and of programming in general. This thesis develops a method of constructing categories of processes upon categories of models with cover systems. As a 2-functor, the construction reduces the task of establishing certain functorial structure in process categories to checking properties of the model categories and chosen cover systems.

The techniques are shown to be applicable to asynchrony. One may move from a synchronous process category to an asynchronous process category by identifying an appropriately structure monad of delay in the model category. Knowing what structure is passed on to the asynchronous setting then translates into checking the interaction between the monad and the relevant structure in the model category.

Finally we consider an approach to programming in process categories based on categories of circuits.

Acknowledgements

First I would like to thank Graham Birtwistle for sparking my interest in functional programming and concurrency theory. I would certainly not have entered graduate studies without his encouragement and support.

I would like to thank my supervisor Robin Cockett for his patience in teaching me category theory, his technical guidance, and his immense devotion to his students.

Thanks to Tom Fukashima, Rob Kremer, Raja Nagarajan, and Conrad Slind for many valuable technical discussions over many bowls of phö. Thanks to Uli Hensel for collaboration and for careful reading of my earlier work. Thanks to my examining committee for carefully reading a draft of this thesis and providing many valuable suggestions. Thanks also to Todd Simpson (and everyone else at Push) for overlooking my absence at work during the preparation of this thesis.

Finally, I would like to thank my family for their support — particularly my wonderful wife Susan for her love and patience over the years. For my children Jeffrey and Erik: you can use the computer now.

Contents

Chapter 1

Introduction

In the world of functional programming, types provide abstract specifications of program behavior. In Standard ML [MTH90], for example, a type can express the fact that a program inputs a pair of lists and returns a list. In languages base on Martin Lof's type theory [Tho91], types are formulae in intuitionistic logic and so one could express the fact that a program takes a pair of lists and returns a list which is their concatenation. In each case, the type system provides the following basic discipline for constructing systems:

$$\frac{f : A \to B \qquad g : B \to C}{f; g \ : \ A \to C}$$

given programs with compatable output and input types, one can safely "plug them together" and infer the type of the composite program. This feature is vital when developing large systems, since one needs the ability to replace components while maintaining the integrity of the entire system.

In concurrent programming, where computation is achieved through the interaction of a collection of autonomous processes, the situation is less satisfactory since the main formalisms (notably CCS [Mil89] and CSP [Hoa85]) are untyped. Approaches to providing type systems (such as *sorts* in the π-calculus [Mil91]) have focused on ensuring that the processes which do interact have a consistent view of the kinds of

values which can be exchanged at their interface. While this is certainly important, it is equally important in plugging systems together that each system has the same view of the interface *over time* — i.e. that they have agreed on a protocol for interaction.

To realize these ideas Abramsky introduced the notion of an *interaction category*. In this setting concurrent systems are viewed as morphisms between typed interfaces, and the type of an interface is a specification of the allowable interaction over time. Such a setting would provide the analogue to typed composition in the functional world, and would facilitate the disciplined construction of concurrent systems.

Abramsky presented interaction categories as a new perspective on the semantics of computation, rather than as an axiomatic framework, and suggested that the logic for these settings be drawn from the ideas of linear logic. To support this new perspective, Abramsky provided a collection of paradigmatic examples: notably SProc [Abr93] and the game-theoretic categories [AJ94].

This thesis is an investigation of a particular method of constructing interaction categories: as a span construction on a category of process models. I take the view that the resulting interaction category (which I call a *process category*) is best understood at the level of the underlying model category. From this perspective, I investigate how to obtain aspects such as asynchrony and linear type structure in a process category.

1.1 Background and related work

Here we review some of the ideas from concurrency theory which are used explicitly in the thesis.

Models of processes

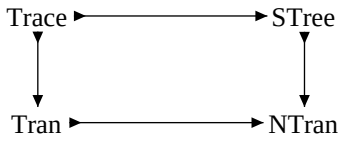
Traditionally, processes have been given an operational interpretation as algebraic structures. In CCS [Mil89], for example, process expressions correspond to nondeterministic labeled transition systems: these are structures (S, Σ, i, δ), where S is a set of states, $i \in S$ is the initial state, Σ is a set of labels, and $\delta \subseteq S \times \Sigma \times S$ is a transition relation. The idea is that the labels represent the channels over which a process can communicate, and the transition relation indicates how the state of the process changes through interaction.

There are a many other models for processes which differ in the aspects of concurrent systems which are considered salient. Transition systems are referred to as "interleaving" models since expressing that actions are independent is achieved through nondeterministic interleaving. In contrast, the so-called "true-concurrency" models (such as event structures [Win87] and pomsets [Pra86]) give an explicit treatment of independence. The models based on game theory not only describe the capabilities of the system (the *player*), but also explicitly represent the capabilities of the environment (the *opponent*). Within these genera there are further distinctions:

- 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 and traces, introduce maximal separation in the state space so that each state determines the history of the process.
- The *branching time* models, such as nondeterministic transition systems and synchronization trees, represent internal choice as nondeterminism. In the *linear time* models, such as traces and deterministic transition systems, the actions available at each state determine the state change of the process.

Sassone, Nielsen and Winskel [SNW93] established relationships in the form of adjunctions amongst many of the standard model categories for concurrency. They show that the behavior models form coreflective subcategories of the associated sys-

tem models, and that the linear-time models are reflective subcategories of the associated branching-time models. For the models of interleaved concurrency, one has the following picture:



where horizontal arrows are reflections and vertical arrows are coreflections. In chapter 4 we show that such relationships between model categories often give rise to even stronger relationships between the process categories. In fact, each of the model categories above gives rise to the same process category.

Bisimulation

State-based models of processes naturally admit the expression of many processes with the same *observable* behavior. Thus in CCS, process expressions are taken to denote transition systems modulo *bisimulation*. Transition systems A and B are bisimulation equivalent if and only if there is a relation $R \subseteq S_A \times S_B$ which satisfies:

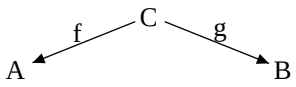
$$(s, s') \in R \ \wedge \ (s, a, t) \in \delta_A \ \Rightarrow \ \exists t'.\, (s', a, t') \in \delta_B \ \wedge \ (t, t') \in R$$
$$(s, s') \in R \ \wedge \ (s', a, t') \in \delta_B \ \Rightarrow \ \exists t.\, (s, a, t) \in \delta_A \ \wedge \ (t, t') \in R$$

A less discerning equivalence is *trace equivalence*: transition systems A and B are trace equivalent if and only if $\mathrm{traces}(A) = \mathrm{traces}(B)$, where $\mathrm{traces}(A) \stackrel{def}{=} \{\ell \mid \exists s.\, (i_A, \ell, s) \in \delta_A^*\}$ and

$$\delta_A^* \stackrel{def}{=} \{(s, [], s) \mid s \in S_A\} \ \cup \ \{(s, a :: \ell, t) \mid \exists u.\, (s, a, u) \in \delta_A \ \wedge \ (u, \ell, t) \in \delta_A^*\}$$

Of course each of the various models of processes comes equipped with it's own notion of behavioral equivalence. Bisimulation in each setting is typically defined in

concrete form, and has lacked a uniform description in the associated model catetories. Joyal, Nielsen and Winskel [JNW93] have proposed that bisimulation be expressed as a stable system of morphisms (a cover system in the terminology of this thesis, although they call it a system of open maps by analogy with the topos theoretic notion) in the model category. The general form is, given cover system $\mathcal{X}$, that objects A and B in a model category are $\mathcal{X}$-bisimilar when related by a span



of morphisms belonging to $\mathcal{X}$. This provides a unifying view of various notions of bisimulation across a range of model categories [JNW93].

For the category of nondeterministic transition systems with fixed label set Σ, the cover system for bisimulation consists of those morphisms which reflect transitions: each cover morphism $f : C \to A$ 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 $f(s') = t$. To see this, note that a span (f, g) between transition systems A and B induces a bisimulation relation

$$\{\, (f(s), g(s)) \mid s \in S_C \,\},$$

and a bisimulation relation on A and B can be seen as (the states of) a transition system with the projections to A and B being cover morphisms. There have been several independent presentations of this particular cover system to describe bisimulation of automata-like structures (for example: Benson [BBS88], Castellani [Cas85], and Van Bentham [VB84]).

Interaction categories

The key idea in Interaction Categories is: rather than viewing processes as *objects* in a model category, one should view processes as *morphisms*. An interaction category

thus has the following general form:

- The objects are interface specifications which express the capabilities and/or responsibilities of concurrent systems over time.
- The morphisms are processes which conform to their interface specifications, and composition of morphisms is given by process interaction.

Examples of interaction categories which have become prominent in programming language semantics are the categories of games and strategies [AJ94, AJM94]. These categories have lead to a solution to the long standing problem of "full abstraction", which strives to reconcile the operational and denotational meanings of higher-order functional programs. Games and strategies also have significance for concurrent programming: A game is a protocol for interaction between a system and its environment, dictating the capabilities and responsibilities of each participant; a strategy is a process which is both deterministic and deadlock-free.

The motivating example of an interaction category, however, is the category SProc of synchronous processes [AGN94]: it's morphisms are (bisimulation equivalence classes of) nondeterministic transition systems, and it's objects are trace specifications. Distilling the presentation of [AGN94], SProc is the following category:

- The objects A are non-empty and prefix-closed sets $S_A \subseteq \Sigma_A^*$ of strings over alphabet Σ_A;
- The morphisms $p : A \to B$ are synchronization trees over alphabet $\Sigma_A \times \Sigma_B$ which satisfy $\mathrm{traces}(p) \subseteq S_A \times S_B$. The synchronization trees over 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 provide a canonical representation of transition systems modulo bisimulation [Acz88].
- 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 $1_A \stackrel{def}{=} \{ ((a,a), 1_{A/a}) \mid a \in S_A \}$, where $A/a \stackrel{def}{=} \{ \ell \mid a :: \ell \in S_A \}$.

Given a process/morphism $A \to B$ of SProc, one can view each state as a relation on $\Sigma_A \times \Sigma_B$: i.e. the set of pairs of actions which the process can exhibit at that state. Indeed, SProc has been likened to a category of "relations extended in time".

Concurrent programming

SProc can be seen as providing a specification-level treatment of concurrent systems. When programming, however, one typically strives to implement systems whose behavior can be predicted — one might like to program concurrent systems as "functions in time".

One such approach is provided by the bicategories of circuits of Katis, Sabadini and Walters (see [KSW94] or [Kat96]): 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 by trace equivalence, obtained through a morphisms of bicategories into Span(**Set**).

A circuit (1-cell) of type $A \to B$ is an automata which at each state transforms inputs in A to outputs in B. Although circuits themselves have extension in time, their types (being simply alphabets) do not.

1.2 Contributions of the thesis

There are two significant contributions conceptually: the first is the view that categories of processes can be constructed in a uniform fashion upon model categories; the second clarifies the nature of asynchrony in process categories. In addition there are several technical results which establish links to other work.

Categories of processes

Certain aspects of the Interaction Category framework remain unsatisfactory, primarily in the formulation of the various examples.

- Examples are presented in concrete form: such presentations have not exposed the general mechanisms which make each example work and thus do not simplify the construction of other examples.
- Examples take canonical representations for processes, thus avoiding the issue of process equivalence.

I have developed the view that process categories are obtained through a span construction upon a model category and a cover system. This approach has certain benefits:

- The construction is modular: the basic mechanism of composition as interaction is given by the construction, and to build a new example one needs only a model category and a cover system. Whether or not the resulting category is of "interest" will of course depend on the particular choice of model category and cover system.
- Process equivalence is treated explicitly as a cover system. This is important since the state-based formalisms which allow finite expression of processes necessarily admit many expressions of the "same" process.

- The construction is 2-functorial, allowing one to reason at the level of the model category in establishing the structure of a process category. This is useful since the logic of process categories is not generally well understood.

As evidence of the latter claim, I have shown that the view of processes as relations is not always appropriate when reasoning about the structure of a process category. In particular, Abramsky's category SProc does not provide a model of the linear exponential type as reported in [AGN94].

Asynchrony in process categories

The use of delay monads is implicit in Robin Milner's construction of asynchrony in process calculus [Mil83], and it has been a common belief that monads can be used to describe asynchrony categorically [AGN94, JNW93]. However, attempts at obtaining asynchrony as a monad construction upon SProc have been unsuccessful [Gay95].

The approach presented in this thesis takes a monad of delay to introduce asynchrony in the model category, and then performs the process construction upon the associated Kleisli category. Since the Kleisli construction does not generally preserve the limits required by the process construction, characterizing the monads which support the construction of processes represents a significant technical development. I believe the techniques developed here offer definite benefits to the formulation of asynchronous process categories:

- Obtaining a well defined notion of composition for asynchronous processes translates into establishing certain properties of the delay monad.
- It exposes at the level of the synchronous model category the technical difficulties in lifting synchronous structure to an asynchronous setting.

For example, the difficulty in lifting the synchronous structure of SProc to ASProc [Gay95] can be seen in the current framework as the difficulty in lifting stable functorial structure over the monad $\_ + 1$ of exceptions.

Although not explicit in their presentation, the categories of games and strategies [AJ94] are inherently asynchronous: at any time a process interacts though at most one of it's interfaces. I present monads of delay for state-based formulations of two models of games: those in which the two players move simultaneously, and those in which the moves of the two players alternate. Obtaining suitable monads in these settings has proven to be quite a delicate business, and the final solution suggests that there are some subtleties hidden in the standard formulation of game-theoretic categories: specifically, certain states are *internal* and receive special treatment with respect to delay.

I conjecture that the process category constructed upon asynchronous interleaved games provides a basis for obtaining the standard categories of games and strategies [AJ94]. The process category upon asynchronous simultaneous games, however, does not appear to have been considered before.

Technical results

I have shown that Abramsky's category SProc of synchronous processes modulo bisimulation arises as a process category upon all of the standard models of interleaved concurrency: from transition systems to trees. Furthermore, SProc modulo trace equivalence is the category of relations on trees. As trees are essentially "sets in time", this result sheds new light on the analogy between SProc and "relations in time". This work is reported in [CS94].

I have provided a reformulation of Abramsky's category ASProc of asynchronous processes as a process category upon the Kleisli category of a delay monad. This provides a functorial analogue of Milner's progression from synchrony to asynchrony in process calculi [Mil83]. As a consequence, I have shown that the cover systems for weak bisimulation and trace equivalence arise through a distributive law for the delay monad. This work appears in [CS95].

I have shown that certain subcategories of SProc — both the partial and total "functions in time" — have an algebraic presentation based on the circuits of Katis, Sabadini and Walters [KSW94]. This work forms a potential basis for programming in process categories and is partially reported in [HS96].

1.3 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 yield 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.

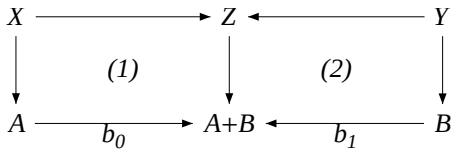
Before concluding, chapter 5 shows that categories of circuits can provide a basis for programming in process categories. Beginning with a simply typed category of partial circuits, one can extract algebraically categories of circuits which satisfy certain safety and liveness properties. These circuits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

1.4 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. If f is an isomorphism, we write f^{-1} as its inverse.

We take $\_\times\_$ to have higher binding precedence than $\_+\_$, 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 category with coproducts is *extensive* if given the following commuting diagram



(1) and (2) are pullbacks if and only if the top row is a pullback. An extensive category with finite limits is called *lextensive*. Note that in an extensive category the coproduct functor is stable (i.e. it preserves pullbacks) and both the injections and

the codiagonal are cartesian (i.e. their naturality squares are pullbacks). Note also that the injections are monic in an extensive category [Coc93].

Chapter 2

Process categories

This chapter presents the general construction of a process category: one begins with a category of models of processes, and then forms a span category quotiented by a cover system. The construction is illustrated using the model category of deterministic transition systems and the cover system for bisimulation, giving rise to Abramsky's category SProc of synchronous processes.

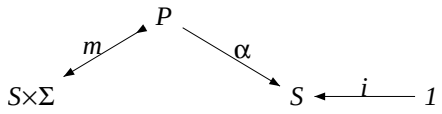
We begin with a brief discussion of model categories presented as sketches. Afterwards we give a treatment of cover systems and develop results which are central to the subsequent development. We then describe the process construction and show that it may be viewed as a 2-functor. This allows one to establish much of the structure of a process category at the level of its model category. To illustrate, we show how one may obtain compact-closure, biproducts and storage in a process category. We conclude by discussing some examples of processes which arise by varying the parameters to the process construction.

2.1 Model categories

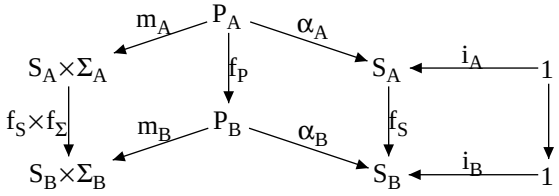
We will generally use the term *model category* to indicate any category which is amenable to the process construction of section 2.3. Typically, however, a model category for processes will be presented as a sketch (see Barr and Wells [BW90] or Wells [Wel94]).

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:*



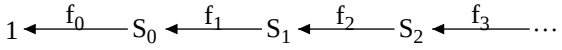
Recall that for a sketch $\mathcal{S}$, the model category $\mathcal{S}(\mathbf{X})$ has as objects diagrams in $\mathbf{X}$ and as morphisms "natural transformations". For example, a morphism $f : A \to B$ in Tran($\mathbf{X}$) corresponds to the following commuting diagram in $\mathbf{X}$:



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 $S \times \Sigma \to S$ specifying 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 $f(\alpha_A(s, a)) = \alpha_B(f(s), f(a))$).

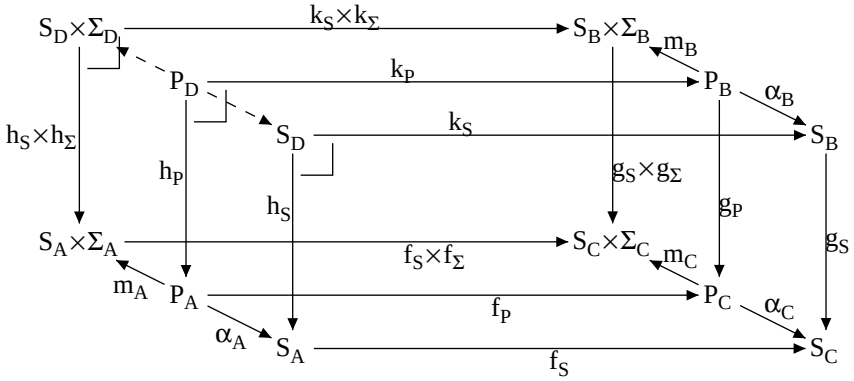
Another useful model category is the category of trees [JM95], which may be constructed upon an arbitrary category $\mathbf{X}$:

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



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 its process category. For the sketches considered in this thesis, the associated model categories will have finite limits which are given pointwise. For example, to form the pullback of morphisms $f : A \to C$ and $g : B \to C$ in Tran($\mathbf{X}$) one forms the pullbacks in $\mathbf{X}$ of the respective components of f and g



(noting that products preserve pullbacks) and obtains the transition system which is the object of the pullback as the induced morphisms shown above.

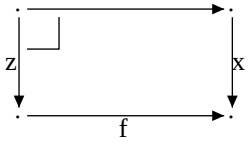
We will also be interested in model categories which have extensive coproducts. These too will be given pointwise when they exist, as is the case in the category of

trees. One obtains a coproduct of transition systems by coalescing 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 provides 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*



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.

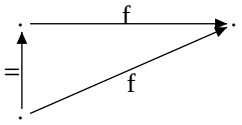
A fact that plays a rather technical role in the subsequent development is that any cover system can be closed by adding all of its right factors, thus obtaining a cover system which is right-factor closed:

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

We will say x is a witness that z is in $\mathcal{X}\downarrow$ when both x and $x; z$ are in $\mathcal{X}$.

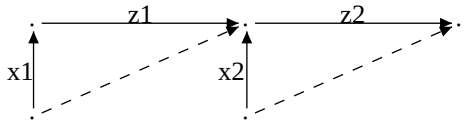
Lemma 2.2.2 *$\mathcal{X}\downarrow$ is a right-factor closed cover system containing $\mathcal{X}$, which inherits left-factor closure from $\mathcal{X}$.*

Proof. Any morphism f in $\mathcal{X}$ belongs to $\mathcal{X}\downarrow$ as witnessed by the identity on the domain of f:

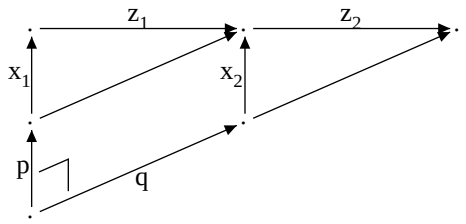


So $\mathcal{X}\downarrow$ contains $\mathcal{X}$ and thus contains all ismorphisms.

To see that $\mathcal{X}\downarrow$ is closed to composition, suppose x_1 and x_2 are witness respectively that z_1 and z_2 belong to $\mathcal{X}\downarrow$:

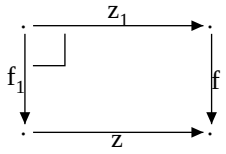


Forming the pullback of $x_1; z_1$ and x_2 yields morphisms p and q which (by the cover system axioms) belong to $\mathcal{X}$.

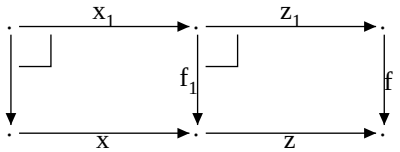


So $p; x_1$ and $p; x_1; z_1; z_2 = q; x_2; z_2$ are in $\mathcal{X}$ by composition, and thus $p; x_1$ is a witness that $z_1; z_2$ belongs to $\mathcal{X}\!\downarrow$.

To see that $\mathcal{X}\!\downarrow$ is closed to pullback, suppose x is witness that z is in $\mathcal{X}\!\downarrow$, and form the pullback of z along an arbitrary morphism f:

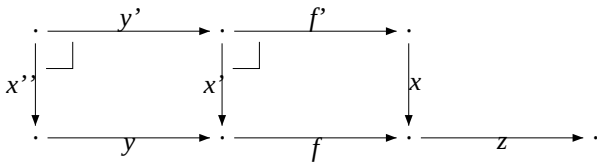


Forming the pullback of x along the obtained morphism f_1 gives a morphism x_1 in $\mathcal{X}$,



and since $x_1; z_1$ is the pullback of $x; z$ along f we also have $x_1; z_1$ in $\mathcal{X}$. Thus z_1 is in $\mathcal{X}\!\downarrow$.

Now suppose $\mathcal{X}$ is left-factor closed, and let z and $f; z$ belong to $\mathcal{X}\!\downarrow$ as witnessed by x and y respectively. To see that f is in $\mathcal{X}\!\downarrow$ consider the following diagram:



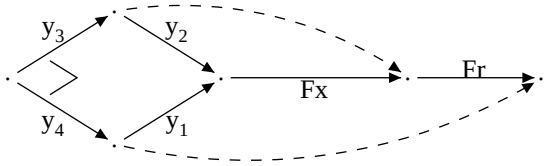
We have x' and x'' in $\mathcal{X}$ since they are pullbacks of x, and $x''; (y; f; z) = y'; f'; x; z$ is in $\mathcal{X}$ by composition. By left-factor closure of $\mathcal{X}$, we have $y'; f'$ in $\mathcal{X}$ and by composition $y'; f'; x = y'; x'; f$ is in $\mathcal{X}$. Thus $y'; x'$ is witness that f is in $\mathcal{X}$. □

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

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:

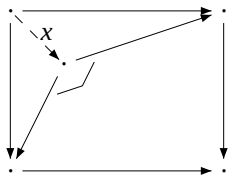
Lemma 2.2.3 *If* $F(\mathcal{X}) \subseteq \mathcal{Y}\downarrow$ *then* $F(\mathcal{X}\downarrow) \subseteq \mathcal{Y}\downarrow$.

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



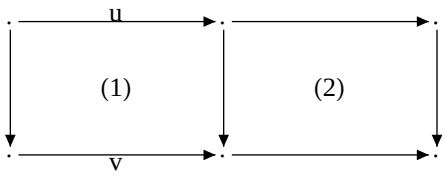
gives y_3 and y_4 in $\mathcal{Y}$. Then $y_4;(y_1;Fx;Fr) = y_3;(y_2;Fx);Fr$ and $y_3;(y_2;Fx)$ are in $\mathcal{Y}$ by composition. Thus Fr is in $\mathcal{Y}\downarrow$. □

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 $\mathcal{X}$.



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* $\mathbf{X}$*:*



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

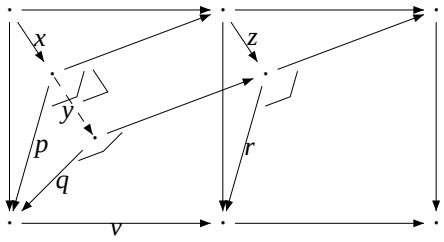
ii) (1) is an $\mathcal{X}$*-pullback whenever (2) is a pullback and the outer square is an* $\mathcal{X}$*-pullback;*

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;

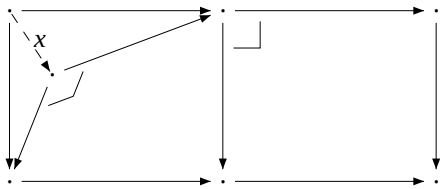
v) if the outer square is an $\mathcal{X}\downarrow$-pullback and u and v belong to $\mathcal{X}\downarrow$ then (2) is an $\mathcal{X}\downarrow$-pullback.

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

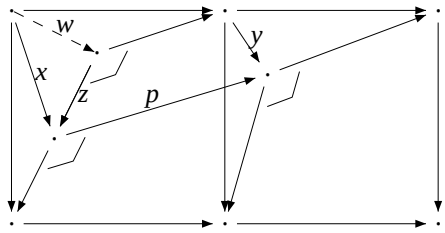


Forming the pullback of r and v, one obtains the pullback inscribed in the entire square; the induced morphism to this pullback is $x;y$. As y is a pullback of z it belongs to $\mathcal{X}$ and thus $x;y$ belongs to $\mathcal{X}$ also.

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

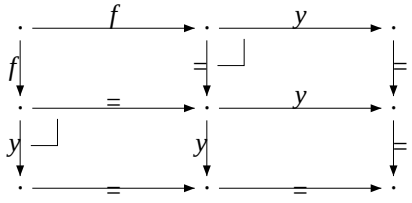


iii) Suppose $\mathcal{X}$ is left-factor closed and let x and y witness that the outer square and (2) are $\mathcal{X}$-pullbacks.



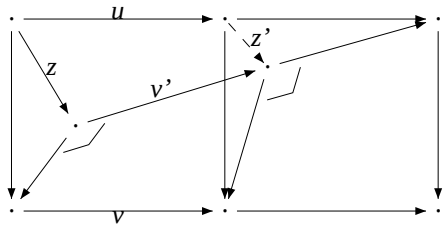
Forming the pullback of y along p gives the pullback inscribed in (1). The induced morphism w then belongs to $\mathcal{X}$ since x and z belong to $\mathcal{X}$.

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) Consider the following diagram:



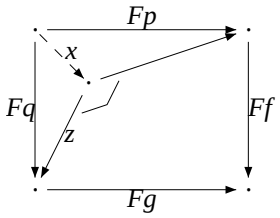
As a pullback of v, we have v' in $\mathcal{X}\!\downarrow$ and thus $z;v'$ is in $\mathcal{X}\!\downarrow$ by composition. Then $z;v' = u;z'$ is in $\mathcal{X}\!\downarrow$ by composition and so z' is in $\mathcal{X}\!\downarrow$ by right-factor closure.

□

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 $F^{\dashv}(\mathcal{X})$ is a cover system on $\mathbf{Y}$ which inherits left-factor and right-factor closure from $\mathcal{X}$.*

Proof. Since 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



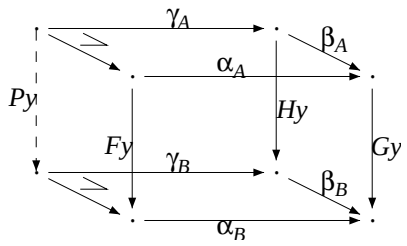
both x and z are in $\mathcal{X}$ and thus Fq is in $\mathcal{X}$ as required.
That $F^{\dashv}(\mathcal{X})$ is left-factor (resp. right-factor) closed whenever $\mathcal{X}$ is left-factor (resp. right-factor) closed is a simple consequence of the fact that F preserves composition. □

As a simple example, note the stable functor from $\mathrm{Tran}(\mathbf{X})$ to $\mathbf{X}$ which takes a transition system to its alphabet: one thus obtains the cover system on $\mathrm{Tran}(\mathbf{X})$ 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:



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 $\mathcal{S}$:

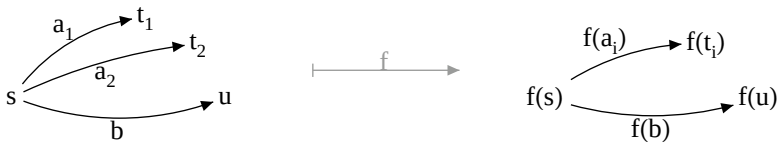
Corollary 2.2.7 *The class of morphisms of $\mathcal{S}(\mathbf{X})$ which are $\mathcal{X}$-cartesian for σ form a cover system; this cover system is left-factor closed iff $\mathcal{X}$ is left-factor closed.*

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

Example 2.2.8 *In* Tran($\mathbf{X}$), *define $\exists^{\mathcal{X}}$ to be the class of morphisms which are $\mathcal{X}$-cartesian for the arrow $m;p_0$ in the sketch of example 2.1.1.*

With respect to transitions, the morphisms of $\exists^{\mathcal{X}}$ are *locally* $\mathcal{X}$ in the following sense: if $f : A \to B$ is in $\exists^{\mathcal{X}}$, then for every state s of A the transitions from 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 state s in A. The following is a simple example of such a morphism:

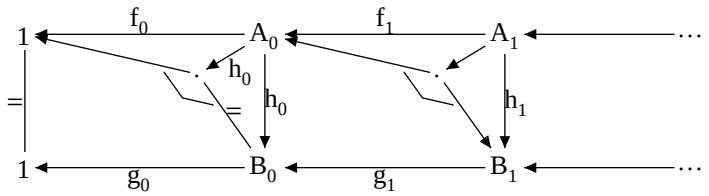


The class of morphisms $\exists^{\mathcal{E}}$ which "reflect" transitions have a history of use in describing bisimulation (see [JNW93]). We will use $\exists^{\mathcal{E}}$ to give bisimulation of processes in SProc, but the classes $\exists^{\mathcal{I}}$ of local isomorphisms and $\exists^{\mathcal{M}}$ of local monics will also be important in the subsequent development.

Cover systems can be similarly obtained in any related category of "transition systems", such as the category of trees:

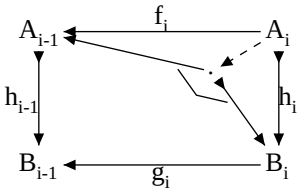
Example 2.2.9 *In* Tree($\mathbf{X}$)*, define* $\exists^{\mathcal{X}}$ *to be the class of morphisms which are* $\mathcal{X}$*-cartsian for every arrow* f_i *in the sketch of example 2.1.2.*

In Tree($\mathbf{X}$), due to the initial state, any morphism h of $\exists^{\mathcal{X}}$ has all of its components h_i in $\mathcal{X}$:



For example, any morphism in $\exists^{\mathcal{E}}$ is an epimorphism and any morphism in $\exists^{\mathcal{I}}$ is an isomorphism. 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.

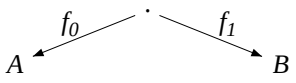
2.3 The process construction

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.

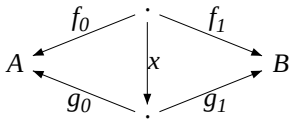
2.3.1 The category of processes

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:

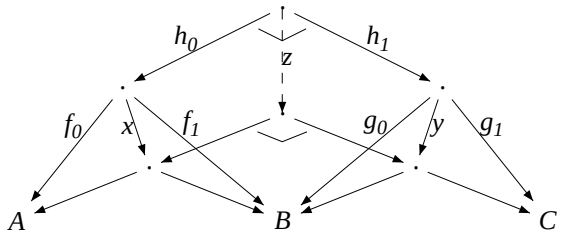
- the objects (or 0-cells) A are those of $\mathbf{X}$
- the hom-category $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})(A,B)$ has as objects (or 1-cells) $f : A \rightsquigarrow B$ those spans (f_0, f_1) in $\mathbf{X}$



with f_0 in $\mathcal{X}_0$ and f_1 in $\mathcal{X}_1$, and as arrows (or 2-cells) $f \Rightarrow g$ the morphisms x of $\mathbf{X}$ for which the following diagram commutes:



- 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 $f \,;; g$ of 1-cells $f : A \rightsquigarrow B$ and $g : B \rightsquigarrow C$ below



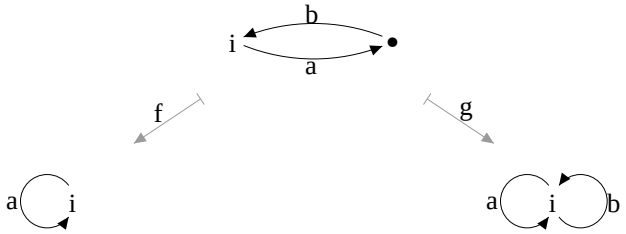
is the span $(h_0; f_0, h_1; g_1)$, and the composite $x \,;; y$ of 2-cells $x : f \Rightarrow f'$ and $y : g \Rightarrow g'$ is the induced 2-cell $z : f \,;; g \Rightarrow f' \,;; g'$ above.

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 $\mathbf{X}$ [Bén67].

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

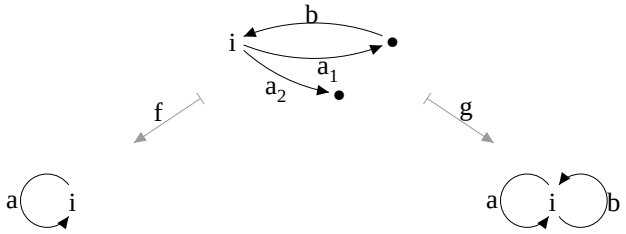
Proof. $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$ results from restricting the hom-categories in the standard bicategory of spans. It is clear that the restricted hom-categories retain the identities and composition. The fact that cover systems are closed under composition and pullback assures that the 1-cell composition also remains well-defined. The fact that cover systems contain all isomorphisms assures that the isomorphisms for associativitiy and unit elimination of 1-cell composition are also retained. □

Viewing a process as a span allows the process implementation (given by the apex of the span) to be abstracted from its interfaces (given by the endpoints of the span). Consider a simple example built upon transition systems



which translates a sequence of a's (i.e. f maps both a and b to a) into an alternating sequence of a's and b's, and may bee seen as a modulo-2 counter.

Note that a process constructed upon deterministic transition systems can be nondeterministic due to potential for the span legs to identify distinct transitions in the interface. For example, the following counter may (or may not) reach a state where it no longer responds:



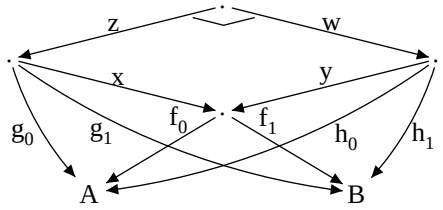
We now consider process equivalence. It is shown in Pare [Par90] that from any bicategory $\mathcal{B}$ one can form a category whose objects are the 0-cells of $\mathcal{B}$ and whose arrows are the 1-cells of $\mathcal{B}$ under horizontal composition. This is done by quotienting 1-cells by the equivalence relation generated by the 2-cells of $\mathcal{B}$. When the hom-categories of $\mathcal{B}$ have pullbacks this amounts to equating a pair of 1-cells exactly when they are related by a span of 2-cells.

For any cover system $\mathcal{X}$ on $\mathbf{X}$, one obtains a sub-bicategory of $\mathrm{Span}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$ by restricting the 2-cells to lie within $\mathcal{X}$. We denote the resulting sub-bicategory

$\mathrm{Span}_{\mathcal{X}}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$.

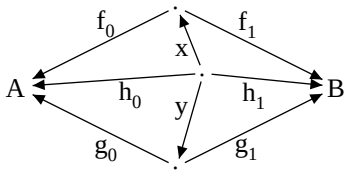
Lemma 2.3.2 $\mathrm{Span}_{\mathcal{X}}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$ *is a bicategory whose hom-categories have pullbacks.*

Proof. The cover system axioms assure that the restricted hom-categories remain categories and retain the isomorphims for the associativity and unit elimination of horizontal composition. The pullback of 2-cells $x : g \to f$ and $y : h \to f$ is given by forming the pullback of morphisms x and y in $\mathbf{X}$:



□

The obtained equivalence, which we will write as $\sim_{\mathcal{X}}$, equates processes f and g exactly when there exists a span (x, y) of 2-cells in $\mathrm{Span}_{\mathcal{X}}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$



Such a span will be called an $\mathcal{X}$-*bisimulation*, and it will be said that the related processes f and g are $\mathcal{X}$-bisimilar. Note that if $\mathcal{X}_0$ (resp. $\mathcal{X}_1$) is left-factor closed then $\mathcal{X}$ is effectively constrained to lie within $\mathcal{X}_0$ (resp $\mathcal{X}_1$). For example, consider the bicategory of partial maps.

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

Definition 2.3.3 $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X},\mathcal{X})$ *is the category which results from quotienting the bicategory* $\mathrm{Span}_{\mathcal{X}}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X})$.

If unspecified, we take $\mathcal{X}_0$ and $\mathcal{X}_1$ to be the entire category $\mathbf{X}$ and take $\mathcal{X}$ to be the intersection $\mathcal{X}_0$ and $\mathcal{X}_1$. In a 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 category of partial maps in $\mathbf{X}$ is given by the process category $\mathrm{Proc}^{\mathcal{M}}(\mathbf{X}, \mathcal{I})$, where $\mathcal{M}$ and $\mathcal{I}$ are the monics and isomorphisms respectively.

Starting with transition systems upon **Set** and taking the cover system for equivalence to be the class $\exists^{\mathcal{E}}$ of local epimorphisms, the resulting process category is (as shown in chapter 4) Abramsky's category SProc of synchronous processes. Thus,

Definition 2.3.4 *For* $\mathbf{E}$ *a regular category, define* $\mathrm{SProc}(\mathbf{E})$ *to be the process category*
$\mathrm{Proc}(\mathrm{Tran}(\mathbf{E}), \exists^{\mathcal{E}})$.

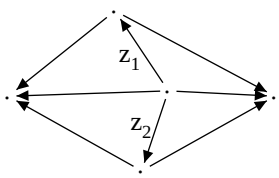
A related process category arises from the model category of trees, and is essentially SProc modulo trace equivalence. We will see in section 2.4 that, with respect to linear types, SProc_T is better structured than SProc.

Definition 2.3.5 *For* $\mathbf{E}$ *a regular category, define* $\mathrm{SProc}_T(\mathbf{E})$ *to be the process category*
$\mathrm{Proc}(\mathrm{Tree}(\mathbf{E}), \mathcal{E})$ — *i.e. the category of relations on the category of trees in* $\mathbf{E}$.

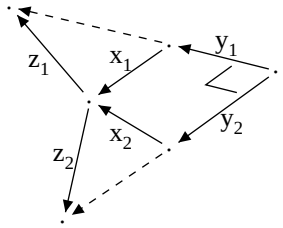
It will be useful to know that the process equivalence induced by a cover system coincides with that induced by its right-closure.

Lemma 2.3.6 $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X}, \mathcal{X})$ *is equivalent to* $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X}, \mathcal{X}\downarrow)$.

Proof. Any $\mathcal{X}$-bisimulation is an $\mathcal{X}\downarrow$-bisimulation since $\mathcal{X} \subseteq \mathcal{X}\downarrow$. Conversely, suppose there is an $\mathcal{X}\downarrow$-bisimulation given by 2-cells z_1 and z_2



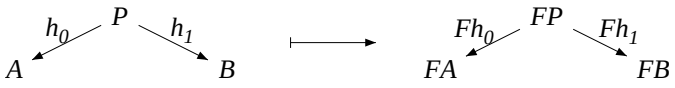
Let x_1 and x_2 be the morphisms of $\mathcal{X}$ which witness the fact that z_1 and z_2 are in $\mathcal{X}\downarrow$. Pulling back x_1 and x_2 gives morphisms y_1 and y_2 in $\mathcal{X}$



and thus an $\mathcal{X}$-bisimulation given by $y_1; x_1; z_1$ and $y_2; x_2; z_2$. □

2.3.2 Lifting functors and natural transformations

Once a category of processes is obtained, one would like to know what structure is present. Often the structure of a process category can be seen to arise directly from structure in its model category. For example, a functor $F : \mathbf{X} \to \mathbf{Y}$ between model categories induces a functor $\mathrm{Proc}(F)$



between the associated process categories under the following conditions:

Lemma 2.3.7 $\mathrm{Proc}(F) : \mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X},\mathcal{X}) \to \mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y},\mathcal{Y})$ *is a functor provided that :*

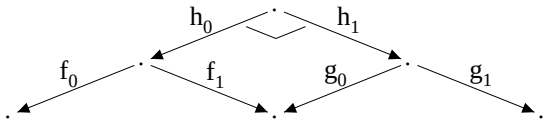
(1) $F(\mathcal{X}_0) \subseteq \mathcal{Y}_0$ *and* $F(\mathcal{X}_1) \subseteq \mathcal{Y}_1$,

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

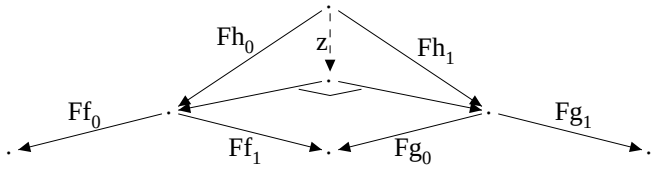
(3) if $h; f = k; g$ *is a pullback in* $\mathbf{X}$ *with* $f \in \mathcal{X}_1$ *and* $g \in \mathcal{X}_0$ *then* $F(h); F(f) = F(k); F(g)$ *is an* $\mathcal{Y}\downarrow$*-pullback in* $\mathbf{Y}$.

Proof. Condition 1 ensures that $\mathrm{Proc}(F)$ produces morphisms in $\mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y},\mathcal{Y})$. Condition 2 ensures that $\mathrm{Proc}(F)$ preserves equality, since every $\mathcal{X}$-bismimulation

will be taken to a $\mathcal{Y}\downarrow$-bisimulation. $\mathrm{Proc}(F)$ will certainly preserve identities since F is a functor. Finally, consider the composition of processes f and g in $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X},\mathcal{X})$



Condition 3 (and 1) ensure that $\mathrm{Proc}(F)(f;g)$ is $\mathcal{Y}\downarrow$-bisimilar (via the induced morphism z)



to the composition of $\mathrm{Proc}(F)(f)$ and $\mathrm{Proc}(F)(g)$: □

For most of the process categories we will consider, the morphisms taken as span legs are unrestricted. In such cases we write $F : (\mathbf{X},\mathcal{X}) \to (\mathbf{Y},\mathcal{Y})$ to indicate that $\mathrm{Proc}(F)$ is a functor $\mathrm{Proc}(\mathbf{X},\mathcal{X}) \to \mathrm{Proc}(\mathbf{Y},\mathcal{Y})$, and note the following simplification of lemma 2.3.7.

Corollary 2.3.8 *$F : (\mathbf{X},\mathcal{X}) \to (\mathbf{Y},\mathcal{Y})$ provided $F(\mathcal{X}) \subseteq \mathcal{Y}\downarrow$ and F takes pullbacks in $\mathbf{X}$ to $\mathcal{Y}\downarrow$-pullbacks in $\mathbf{Y}$.*

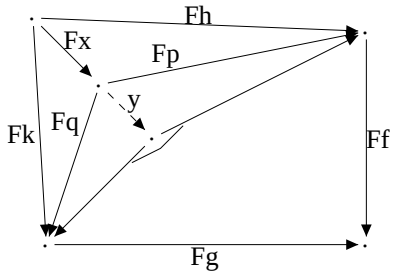
Equivalently, the functors which lift in the simple case are those which preserve cover pullbacks.

Lemma 2.3.9 *F takes $\mathcal{X}$-pullbacks to $\mathcal{Y}$-pullbacks if and only if $F(\mathcal{X}) \subseteq \mathcal{Y}$ and F takes pullbacks to $\mathcal{Y}$-pullbacks.*

Proof. Suppose F takes $\mathcal{X}$-pullbacks to $\mathcal{Y}$-pullbacks. Pullbacks are $\mathcal{X}$-pullbacks and thus taken to $\mathcal{Y}$-pullbacks. Furthermore, the $\mathcal{X}$-pullback $x;1 = x;1$ is taken to the

$\mathcal{Y}$-pullback $Fx;1 = Fx;1$, so $Fx \in \mathcal{Y}$.

Conversely, suppose $h;f = k;g$ is an $\mathcal{X}$-pullback and that $x \in \mathcal{X}$ is the induced morphism to the inscribed pullback $p;f = q;g$. Since F takes pullbacks to $\mathcal{Y}$-pullbacks, the induced morphism y

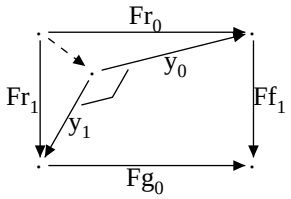


to the pullback inscribed in $F(p);F(f) = F(q);F(g)$ is in $\mathcal{Y}$. Then $Fx;y$, the unique morphism to the pullback inscribed in $F(h);F(f) = F(k);F(g)$, is in $\mathcal{Y}$ as required. □

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:

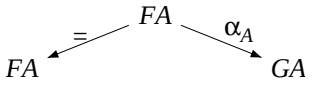
Lemma 2.3.10 *Let $\mathcal{Y}_0$ and $\mathcal{Y}_1$ be left-factor closed. If $F(\mathcal{X}_0) \subseteq \mathcal{Y}_0$, $F(\mathcal{X}_1) \subseteq \mathcal{Y}_1$ and $F(\mathcal{X}) \subseteq \mathcal{Y}_0 \cap \mathcal{Y}_1$ then* $\mathrm{Proc}(F)$ *is a functor* $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X},\mathcal{X}) \to \mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y})$.

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



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

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 $\mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X},\mathcal{X}) \to \mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y},\mathcal{Y})$:

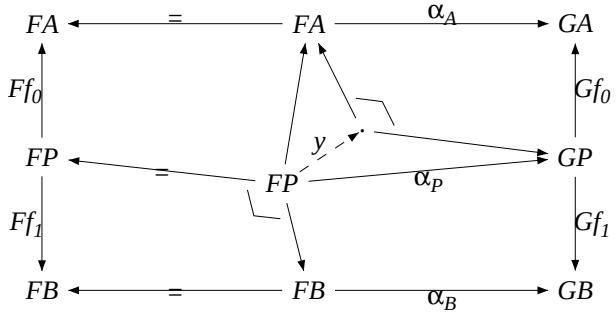


Lemma 2.3.11 $\mathrm{Proc}(\alpha)$ *is a natural transformation* $\mathrm{Proc}(F) \Rightarrow \mathrm{Proc}(G)$ *whenever* :

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

(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 $\mathbf{Y}$:

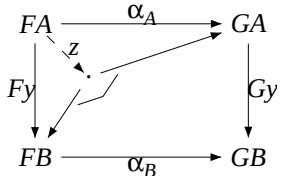


To commute in the process category $\mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y},\mathcal{Y})$ it is sufficient that induced morphism y belongs to $\mathcal{Y}\!\downarrow$. □

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

Lemma 2.3.12 *If* $\mathcal{Y}$ *is left-factor closed, then* $\alpha : F \Rightarrow G$ *is* $\mathcal{Y}$*-cartesian whenever* α_A *is in* $\mathcal{Y}$ *for all* A *in* $\mathbf{X}$.

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



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

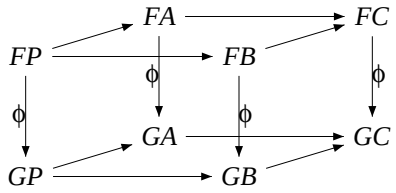
The following proves useful for identifying functorial structure which lies in the domain of the process construction.

Lemma 2.3.13 *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



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. □

2.3.3 The 2-functor Proc

We can thus view the process construction as a 2-functor, with domain the 2-category **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 2.3.7,
- 2-cells – natural transformations $\alpha : F \Rightarrow G$ which satisfy lemma 2.3.11.

Proposition 2.3.14 Proc : **PModel** $\to$ **Cat** *is a 2-functor which preserves products.*

Proof. It is clear that Proc preserves composition of 1-cells (i.e. $\mathrm{Proc}(F) ;; \mathrm{Proc}(G)$ is $\mathrm{Proc}(F ;; G)$) and vertical composition of 2-cells (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

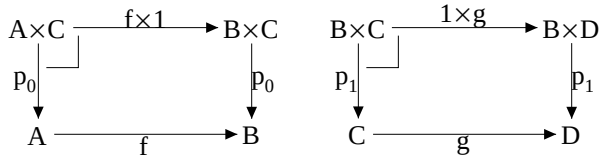
One of the tenets of this approach to constructing process categories is that functorial structure on a process category can be predicted from related structure in its model category. Here we consider how this approach may be used in obtaining a categorical model of Linear Logic [Gir87] — i.e. a *linear category*. Specifically, we consider how to obtain compact-closure, biproducts and storage in a process category such as SProc.

2.4.1 Compact-closure

A $*$-autonomous category [Bar79] provides a model of the multiplicative constructs of linear logic [Bar91, See89]. A compact-closed category is a $*$-autonomous category in which the tensor and cotensor coincide. In this case the complementation morphisms $\tau : \top \to A \otimes A^{\perp}$ allow a form of feedback, so that processes may be connected in

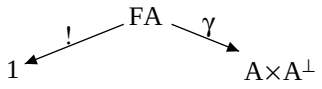
cycles. For example, given a process $f : A \to A$ one can form the "closed loop" $\tau_A; f \otimes 1; \tau_{A^\perp}^\perp : \top \to \top$

In a process category which is self-dual, compact-closure can arise from the product in its model category. Products in any category preserve pullbacks and also preserve any cover system since the following squares

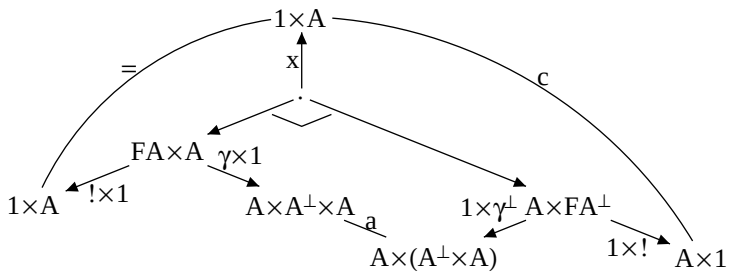


are always pullbacks. Furthermore, the natural isomorphisms for associativity, symmetry and unit elimination are cartesian. Thus the product and the associated natural transformations in a model category always induce (via the 2-functor Proc) a symmetric tensor on the process category – although there it is not generally a product.

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 a weakly distributive category with the weak distribution δ given by associativity [CS92]. To further obtain a compact closed category is sufficient to have, for each A, a span τ_A



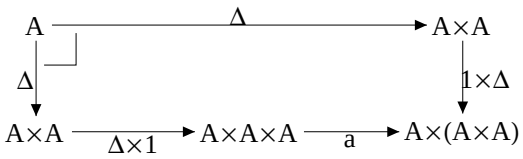
and a morphism x of $\mathcal{X}$ for which:



Note that the above diagram is simply a bisimulation witnessing the process equation $c = 1 \otimes \tau_A; \delta; \tau_A^{\perp} \otimes 1 : I \otimes A \to A \otimes I$.

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. Noting that the following



is a pullback, the cover morphism x is given by the unit elimination isomorphism $u : 1 \times A \to A$. □

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

2.4.2 Biproducts

Biproducts (i.e. products which are simultaneously coproducts) can be used to model the additive constructs of linear logic. As shown by Abramsky, biproducts allow the implementation of nondeterministic choice in SProc: given processes $f, g : A \to B$, the composite process $\Delta; f \otimes g; \nabla : A \to B$ behaves either as f or as g nondeterministically.

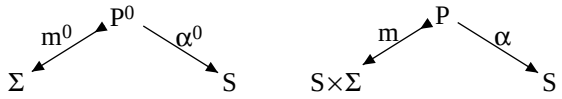
Lindner [Lin76] observed that the span category of any lextensive category has biproducts: since the coproduct functor is stable and the associated natural transformations are cartesian, the span category has finite coproducts which are simultaneously products since the span category is self dual. To obtain biproducts in a process

category it is then sufficient that coproducts in the model category preserve the cover system.

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

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*



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

The exponential type (or storage) of linear logic can be modelled in a $*$-autonomous category by a comonad $!A$ together with a commutative comonoid on $!A$ [Bar91, See89]. This structure allows one to view processes as resources which can be replicated (through contraction $!A \rightarrow !A \otimes !A$) or discarded (through weakening $!A \rightarrow \top$). Storage is also necessary to encode the lambda calculus in linear logic, so that one may combine both functional and concurrent programming in the same framework.

A common method of obtaining the exponential type is to use the Fock construction [Bar91, BPG94]: in a compact-closed category with finite biproducts, define $!A \stackrel{def}{=} \Sigma_{n \in \omega} \otimes^n_s A$, 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 $\otimes^n A$

$$\otimes^n A \underset{\sigma_{n!}}{\overset{\sigma_1}{\rightrightarrows}} \otimes^n A \xrightarrow{\quad q \quad} \otimes^n_s A$$

In the category Rel of sets and relations, 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**:

$$A^n \underset{\sigma_{n!}}{\overset{\sigma_1}{\rightrightarrows}} A^n \xrightarrow{\quad q \quad} M_n(A)$$

Given any $r : \otimes^n A \rightarrow B$ in Rel which satisfies $\forall \sigma \in \mathcal{S}_n.\ \sigma; r = r$, the mediating morphism $\otimes^n_s A \rightarrow B$ 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 $(M f_0, M f_1)$.

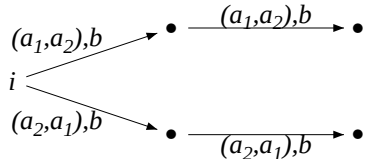
Storage in SProc

Unfortunately the Fock construction fails in SProc, contrary to suggestions in [AGN94]. 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 $q : A \times A \rightarrow M_2(A)$ of

the identity and symmetry morphisms in the category of traces gives the obvious candidate for the coequalizer in SProc: the process

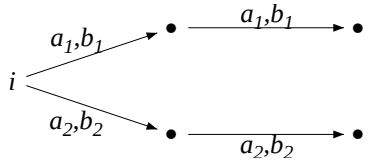
$$q \stackrel{def}{=} 1_{A\otimes A}[((a_1,a_2),(a_1,a_2)) \mapsto ((a_1,a_2),\{|a_1,a_2|\})]$$

using the notation of [AGN94]. The process $r : A \otimes A \to B$ given by the transition system



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.

In fact, $!(\_)$ fails to preserve composition in SProc and is thus not a functor. 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



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

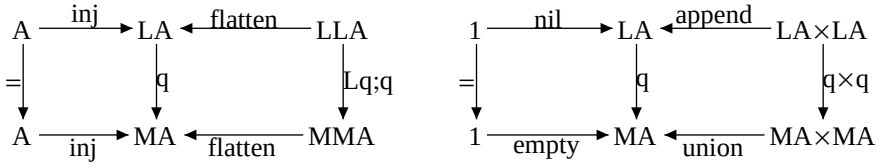
The above failing corresponds to the fact that the multiset functor is not an $\exists^{\mathcal{E}}$-stable functor on the model category for SProc. It does not, however, refute the possibility that $!(\_)$ is a functor under a more generous equivalence. As we will see, $!(\_)$ becomes a functor if we quotient SProc by trace equivalence rather than bisimulation.

Storage via multisets

Since a process category $\mathrm{Proc}(\mathbf{X},\mathcal{X})$ is self dual, any monad there is simultaneously a comonad and any monoid is simultaneously a comonoid. An obvious approach to

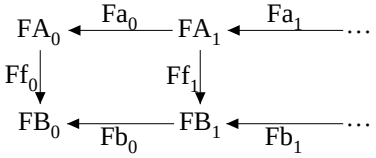
obtaining an interpretation of !(_) in a process category is thus to lift a monad with a commutative monoid structure through the process construction.

In the category **Set**, both the list functor L and the multiset functor M have such structure. Furthermore, these structures are also related by a natural transformation q,



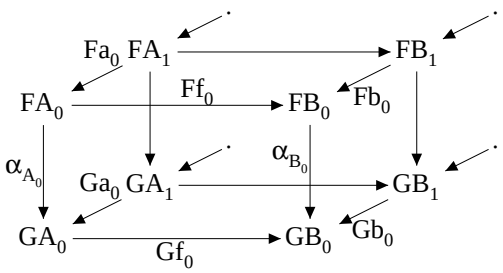
(given by the sum $\Sigma_{n\in\omega} q_n$ of coequalizers of $\mathcal{S}_n$ — recalling that $L(A) \stackrel{def}{=} \Sigma_{n\in\omega}(\times^n)A$) which is pointwise a regular epimorphism and is also $\mathcal{E}$-cartesian. Since L is a stable functor and its associated natural transformations are cartesian, by lemma 2.3.13, M is an $\mathcal{E}$-stable functor and its associated natural transformations are $\mathcal{E}$-cartesian. Thus the monad and monoid structure on multisets lifts through the process construction to the category of relations. In addition, the isomorphisms $MA{\times}MB \to M(A{+}B)$ and $1 \to M0$ in **Set** induce the isomorphisms $!A{\otimes}!B \to !(A{\times}B)$ and $\top \to !1$ in Rel required to make !(_) a model of the linear exponential type.

In the category of trees, any functor $F : \mathbf{X} \to \mathbf{Y}$ induces a functor $\mathrm{Tree}(F) : \mathrm{Tree}(\mathbf{X}) \to \mathrm{Tree}(\mathbf{Y})$ defined pointwise



and any natural transformation $\alpha : F \Rightarrow G : \mathbf{X} \to \mathbf{Y}$ induces a natural transformation

$\mathrm{Tree}(\alpha) : \mathrm{Tree}(F) \Rightarrow \mathrm{Tree}(G)$ given by the family $\{(\alpha_{A_0}, \alpha_{A_1}, \ldots) \mid A \in \mathrm{Tree}(\mathbf{Y})\}$:



The topos $\mathrm{Tree}(\mathbf{Set})$ [JM95] in this way inherits the relationship between the list and multiset monads from **Set**, making $\mathrm{Tree}(M)$ and its associated structure $\mathcal{E}$-cartesian. The isomorphisms $1 \to \mathrm{Tree}(M)(0)$ and $\mathrm{Tree}(M)(A) \times \mathrm{Tree}(M)(B) \to \mathrm{Tree}(M)(A+B)$ are also inherited and thus the process category $\mathrm{Proc}(\mathrm{Tree}(\mathbf{Set}), \mathcal{E})$ — i.e. the category of relations on trees or alternatively SProc modulo trace equivalence — provides a model of the linear exponential type.

Proposition 2.4.5 SProc_T *is a linear category.*

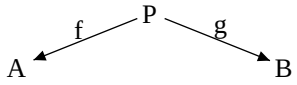
Note that the multiset functor exists in any locos in which each object can be totally ordered [Coc90], so this construction may be performed in a more general setting.

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



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

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

where $\exists^{\mathcal{R}}$ are the local retractions. Note that the cover system for equivalence is actually forced to be the local isomorphisms $\exists^{\mathcal{I}}$: 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 [Mil89].

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:

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

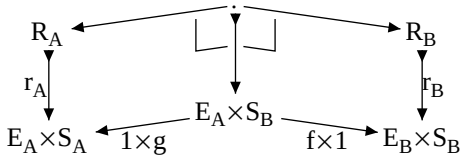
We will see later that this process category is essentially the category of trees in

X, and that the former category is essentially the category of partial maps on trees.

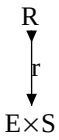
2.5.2 Chu spaces

The category of Chu spaces (or Chu(**Set**, 2)) advocated by Pratt [Pra95] 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 **Set**.

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



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

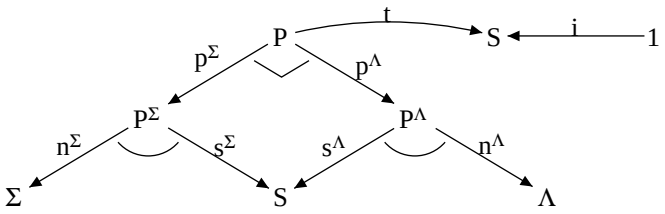


The cover system for left legs is given by the morphisms which are cartesian for r and have their E-component an isomorphism, while the cover system for right legs are the morphisms which are cartesian for r and have their S-component an isomorphism. The cover system for equivalence is then forced to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

2.5.3 Simultaneous games

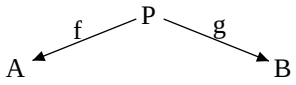
Consider the form of 2-player game in which both players move simultaneously. Such games can be modelled as transition systems which are labelled by pairs taken from two independent alphabets.

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



Here the morphisms $\langle s^\Sigma, n^\Sigma \rangle : P^\Sigma \to S \times \Sigma$ and $\langle s^\Lambda, n^\Lambda \rangle : P^\Lambda \to S \times \Lambda$ indicate the permissible actions for player and opponent respectively. P represents the collection of all permissible pairs of actions by player and opponent, and t determines how each pair of independent actions effects the state of the game. Note that such games are effectively transition systems with states S, labels $\Sigma \times \Lambda$, and permission set P.

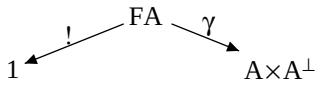
Consider as cover systems for span construction the classes of morphisms $\mathcal{L}_0$ and $\mathcal{L}_1$ which are cartesian for the arrows s^Σ and s^Λ respectively. A process



constructed in this manner has the property that at any state s in P, each Σ-move permissible at state fs in A has a unique corresponding Σ-move permitted at s and each Λ-move permissible at state gs in B has a unique corresponding Λ-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



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.

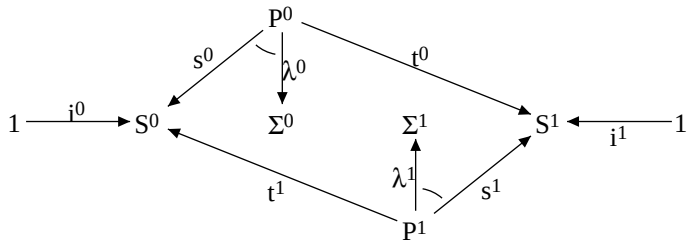
Another disappointing note is that these process categories do not have biproducts since the coproduct injections do not belong to the cover systems chosen for span construction.

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 [AJM94].

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:*



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.

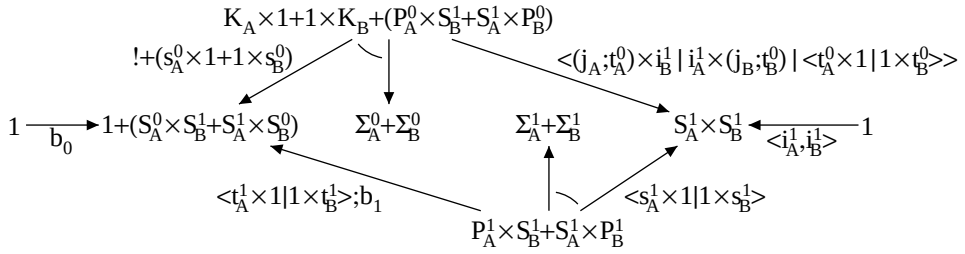
There is an involution $\_^{\perp}$ on the model category which switches the roles of player and opponent, and taking $\mathcal{L}$ to be the class of morphisms which are cartesian for s^0 we obtain the process category

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

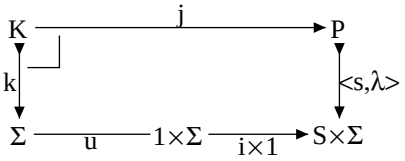
of games and synchronous strategies. Like the process category of section 2.5.3, the morphisms of Strat($\mathbf{X}$) can be seen as deterministic and deadlock-free processes.

The process category 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 Game($\mathbf{X}$),



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 $p_0 + p_0$; $\lambda_A^1 + \lambda_B^1$, and the initial player transitions of each game are extracted as follows:



The unit for the tensor is given by the final object in Game($\mathbf{X}$). Note that although $\_ \otimes \_$ is an associative bifunctor on the model category 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 section 4.1).

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,* 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 [AJ94]. In particular, implementing the complementation process $I \to A \oplus A^{\perp}$ necessary for $*$-autonomy requires asynchrony.

2.6 Summary

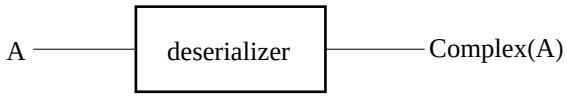
We have presented the construction of a process category as a category of generalized relations: given a category of models of processes and a cover system for behavioral equivalence, a category of processes is formed as a span category quotiented by the cover system. Choosing additional cover systems 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 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

Asynchronous process categories

Consider a process which translates a sequence of atomic values at one interface into a sequence of complex data structures the other interface:



Such a process must delay producing an output until the appropriate number of inputs has been received, and is thus asynchronous.

This chapter presents an approach to building categories of asynchronous processes: one begins with a model category for synchronous processes and a monad which adds delay to the models; the process construction is then performed upon the Kleisli category of the delay monad. This approach, which stems naturally from the desire to calculate at the level of the model category, raises several technical questions. When does a Kleisli category have enough limits to support the process construction? How does one obtain a cover system which "ignores" delay? When is the functorial structure of the synchronous process category preserved in the asynchronous setting?

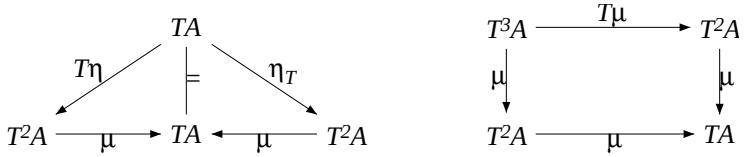
We begin with a review of the Kleisli construction and the conditions under which the structure of the underlying category can be lifted to a Kleisli category. We then isolate a class of monads whose Kleisli categories have finite limits, and develop

conditions under which these Kleisli categories inherit the cover-stable structure of the underlying category. The lifting of stable functors provides an important 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 also consider asynchrony in the context of games, both simultaneous and interleaved, where the notion of delay is more intricate.

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:



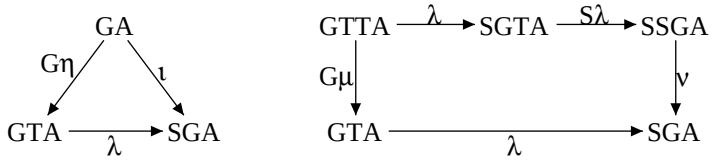
A standard example is the monad of exceptions: if $\mathbf{X}$ is a category with finite coproducts and X is an object of $\mathbf{X}$, then $(\_ + X,\ b_0,\ a; 1+\nabla)$ is a monad.

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 $\mathbf{X}$.

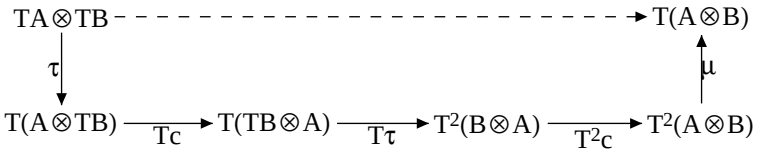
Suppose that (T, η, μ) and (S, ι, ν) are monads on categories $\mathbf{X}$ and $\mathbf{Y}$ respectively.

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



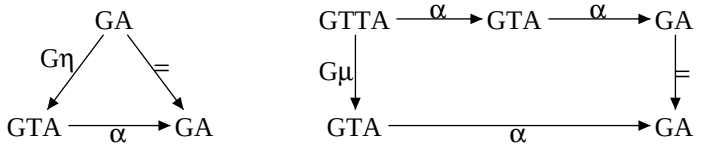
commutes for all A in $\mathbf{X}$. Such a λ is called a *distribution* for G over T and S.

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 [Coc93], a distribution for the functor $\_{\times}Y$ over the monad $\_{+}X$ is given by the morphism $d; 1{\times}p_1 : (A{+}X){\times}Y \to A{\times}Y + X$. In each case the distribution for the functor $\_ \otimes Y$ (i.e. the tensor strength) induces a distribution for the functor $\_ \otimes \_$ as follows:



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

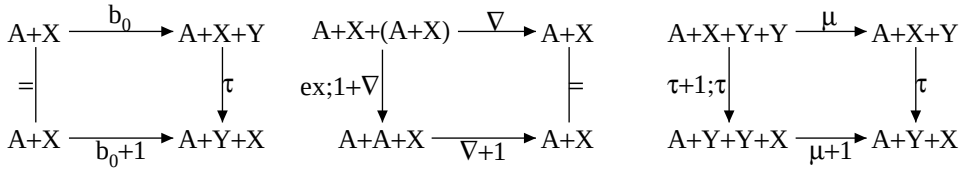


is called a *T-action* for G.

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 codiagonal lift over the monad* $\_+X$*, as do the unit and multiplication of* $\_+X$*:*



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 [Mul94a] and is based on the fact that the 1-cells of Lift($\underline{\mathbf{X}}$) correspond exactly to distributions.

3.2 Stable monads

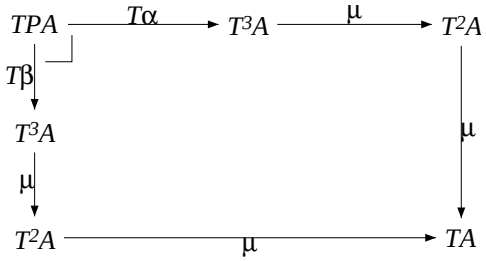
Here we isolate a class of monads (*stable* monads) whose Kleisli categories have pullbacks, and develop conditions under which stable functors and cartesian natural transformations can be lifted over these monads. The latter conditions are refinements on the conditions for lifting fuctors and natural 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 [Kel92]. We say that a club is a *stable monad* when its Kleisli category has pullbacks.

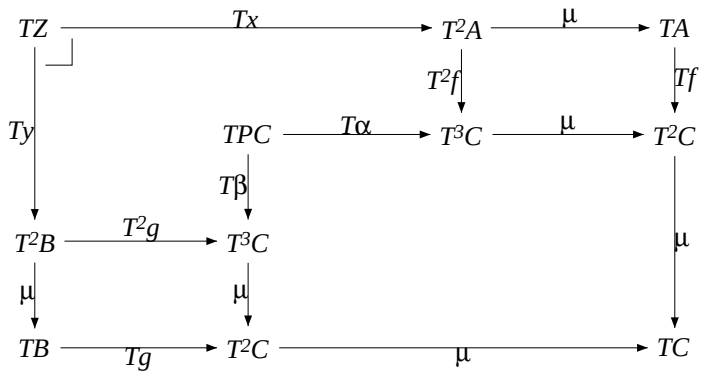
Proposition 3.2.1 *A club (T, η, μ) is a stable monad if and only if there exists a stable functor $P : \mathbf{X} \to \mathbf{X}$ and cartesian natural transformations α and $\beta : P \Rightarrow TT$ which make*



a pullback in $\mathbf{X}$.

Proof. Suppose that $\mathbf{X}_T$ has pullbacks. Note that the counit ϵ_A 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 then 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 limit (x, y) of diagram (Tg, β, α, Tf) makes



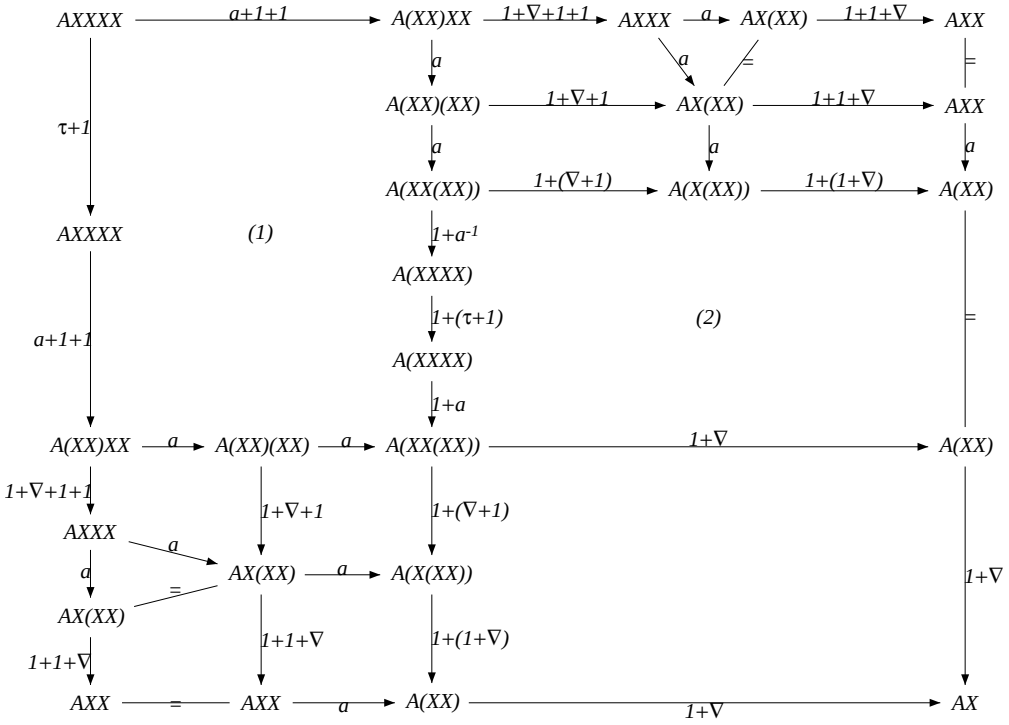
a pullback in $\mathbf{X}$ (as T is stable and μ is cartesian), and thus (x, y) is a pullback of f and g in $\mathbf{X}_T$. □

A fundamental example of a stable monad is the monad $\_+X$. Here the additional structure required by propoosition 3.2.1 is given by the functor T^3 and the natural transformations $T\mu$ and $\tau; T\mu$ respectively, where τ is the isomorphism $A + X + Y \to A + Y + X$. We will refer to monads whose stable structure is given in this way as *exception monads.*

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:



(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 $\mathbf{X}$.*

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

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 $\mathcal{X}$. Note that the functor of a stable monad always reflects a cover system.

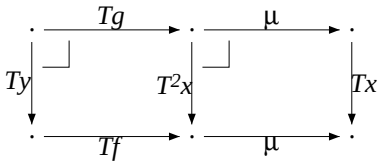
Lemma 3.2.4 *If (T, η, μ) is a club a category $\mathbf{X}$ with a cover system $\mathcal{X}$, then the functor T reflects $\mathcal{X}$.*

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 $\mathcal{X}$. □

Let (T, η, μ) be an $\mathcal{X}$-stable monad on category $\mathbf{X}$. The class of morphisms $F_T(\mathcal{X})$ in $\mathbf{X}_T$ is a cover system on $\mathbf{X}_T$ provided the isomorphisms of $\mathbf{X}_T$ are just the isomorphisms of $\mathbf{X}$ lifted.

Proposition 3.2.5 *If $\mathcal{I}_{\mathbf{X}_T} = F_T(\mathcal{I}_{\mathbf{X}})$ then $F_T(\mathcal{X})$ is a cover system on $\mathbf{X}_T$.*

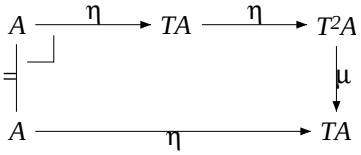
Proof. $\mathcal{X}_T$ is closed under composition as F_T is a functor. To see that $\mathcal{X}_T$ is closed to pulling back along arbitrary morphisms, suppose x is in $\mathcal{X}$ and f is a morphism of $\mathbf{X}_T$. The following pullback in $\mathbf{X}$ corresponds to a pullback $g;F_Tx = F_Ty;f$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



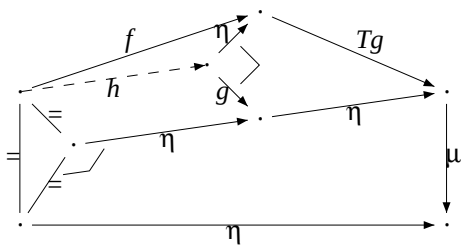
y is in $\mathcal{X}$ as T preserves and reflects $\mathcal{X}$, and thus F_Ty is in $\mathcal{X}_T$. □

Note that this cover system does not generally inherit left-factor closure from $\mathcal{X}$. Of course $F_T(\mathcal{X})$ can always be made into a cover system by adding the additional isomorphisms of $\mathbf{X}_T$ and closing under composition and pullback. In either case, we will call the obtained cover system $\mathcal{X}_T$.

Lemma 3.2.6 $\mathcal{I}_{\mathbf{X}_T} = F_T(\mathcal{I}_{\mathbf{X}})$ *whenever η is monic and the following is a pullback for all A in $\mathbf{X}$:*



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 $f = F_T(h)$:

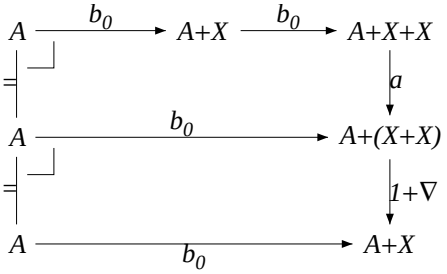


Similarly, there exists k such that $g = F_T(k)$. Furthermore, $h;k = 1$ and $k;h = 1$ as η (i.e. the coproduct injection) is monic. □

The exception monad $\_ + X$ (and thus the derived monads of delay on transition systems) have the abovementioned property.

Example 3.2.7 *In a lextensive category,* $\mathcal{I}_{\mathbf{X}_{\_+X}} = F_{\_+X}(\mathcal{I})$ *whenever the coproduct functor preserves the cover system* $\mathcal{X}$.

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



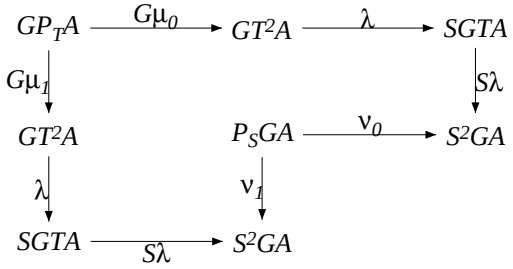
□

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

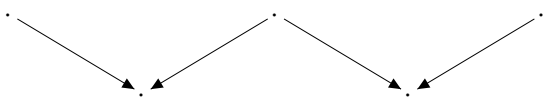
Suppose now that T and S are cover-stable monads on $(\mathbf{X}, \mathcal{X})$ and $(\mathbf{Y}, \mathcal{Y})$ respectively, and that $G : (\mathbf{X}, \mathcal{X}) \to (\mathbf{Y}, \mathcal{Y})$ is a cover-stable functor.

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*



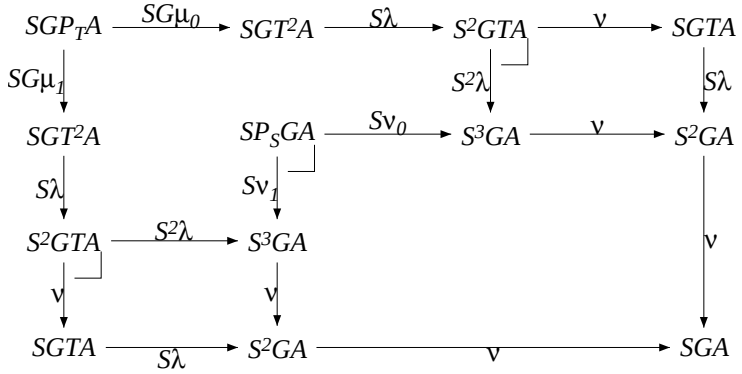
is a $\mathcal{Y}$-double pullback for all A in $\mathbf{X}$.

Here we use the term *double-pullback* to indicate the limit of the diagram



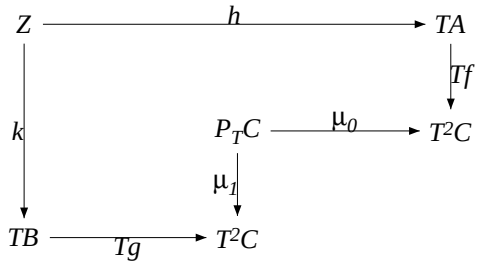
which, of course, may be formed by a combination of pullbacks.

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

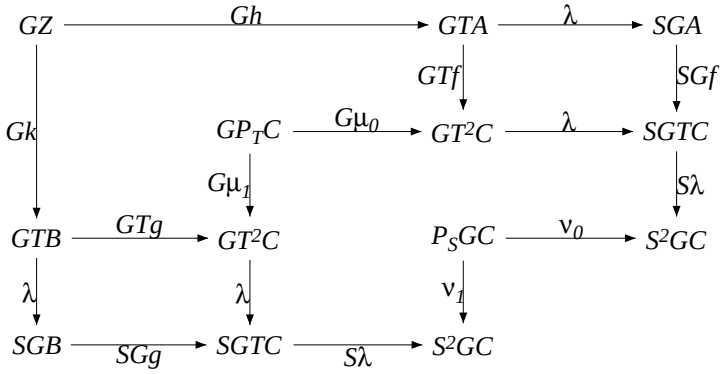


Since S reflects cover-pullbacks, the diagram in question is a $\mathcal{Y}$-double pullback.

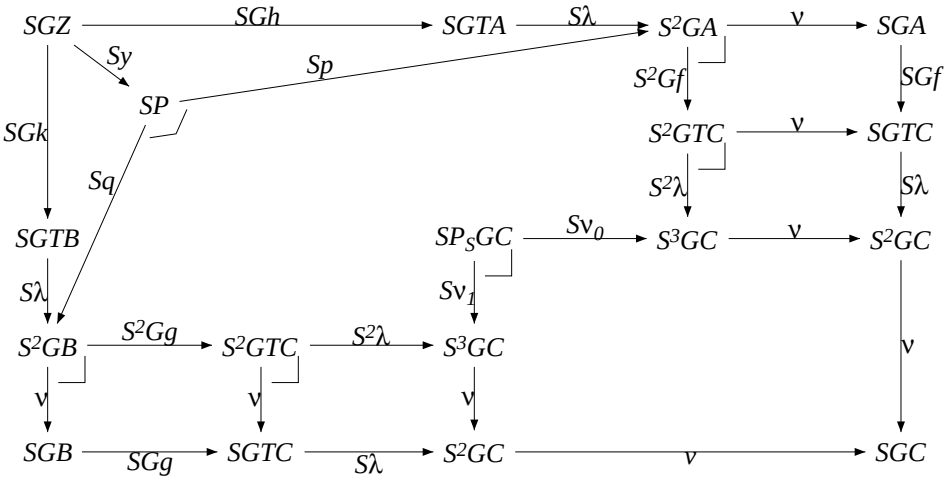
($\Leftarrow$) If $h; f = k; g$ is an $\mathcal{X}_T$-pullback then as U_T preserves and T reflects cover pullbacks the following is an $\mathcal{X}$-double pullback:



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



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



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

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

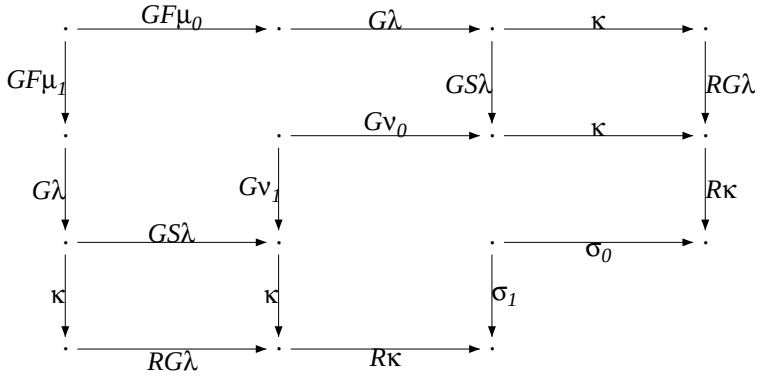
$$Gk; \lambda = y; q = y; \iota; Sq; \nu$$

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

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 $(\mathbf{X}_T, \mathcal{X}_T) \to (\mathbf{Z}_R, \mathcal{Z}_R)$.*

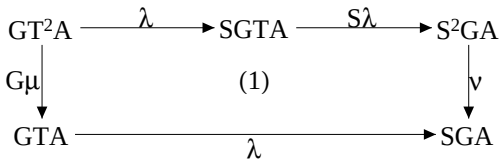
Proof. The required diagram is as follows:



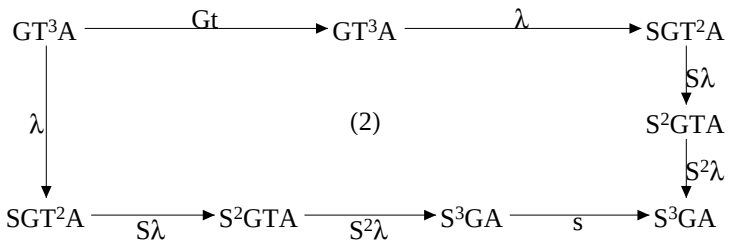
□

The following lemma is useful for deciding whether or not a distribution involving exception monads is cover stable. In the following lemma, let S and T be stable monads whose additional structure (c.f. proposition 3.2.1) arises from isomorphisms $t : T^2 \Rightarrow T^2$ and $s : S^2 \Rightarrow S^2$ respectively.

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

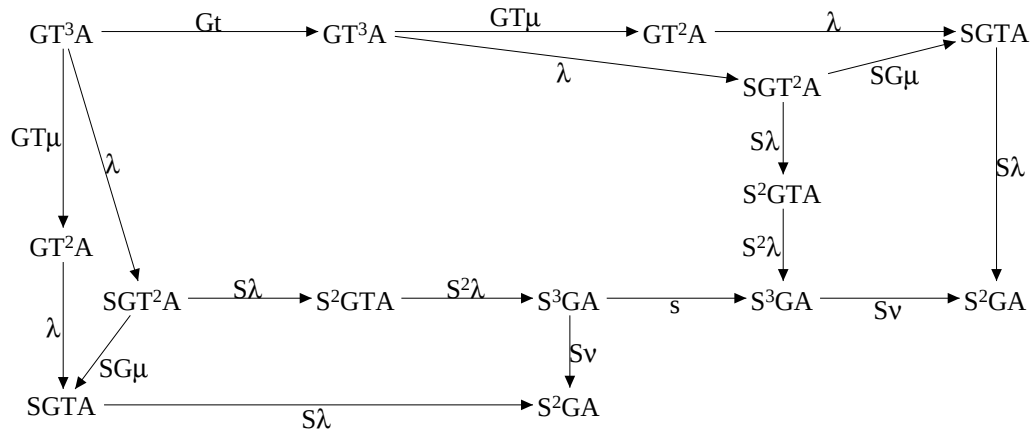


is a pullback, then λ is cover-stable if and only if



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

Proof. Since (1) is a pullback, the $\mathcal{Y}$-double pullback of proposition 3.2.8 must be constructed as follows:



□

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 $\tau+1;\tau$ is a stable distribution of $\_+Y+Y$ over $\_+X$.*

Proof. Concerning the diagrams of lemma 3.2.10, (1) commutes by coherence for symmetric monoidal categories, and both (1) and (2) are pullbacks since τ is an isomorphism. □

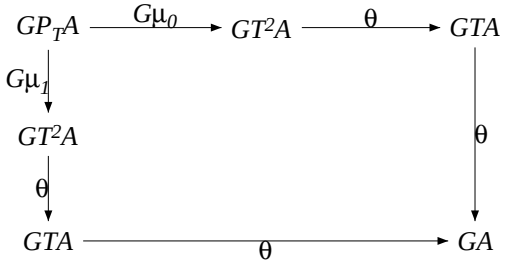
Unfortunately, most other examples of distributions over $\_+X$ are not cover stable: Both the product and coproduct functors have diagram (1) of lemma 3.2.10 a pullback, but (2) is neither a pullback nor an $\mathcal{R}$-pullback since the inscribed pullback is "larger".

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:

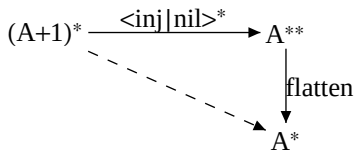
Corollary 3.2.12 *If $\alpha : GT \Rightarrow G$ is a T-action for G, then G_α is functor $\mathbf{X}_T \to (\mathbf{Y}, \mathcal{Y})$ if and only if*



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.

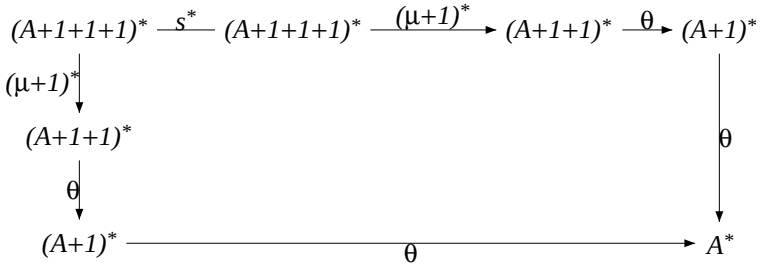
An example relevant to obtaining a cover system for weak bisimulation is given in **Set** by the monad of lists ($\_^*$, inj : $Id \Rightarrow \_^*$, flatten : $\_^{**} \Rightarrow \_^*$) and the natural transformation $\theta : (\_ + 1)^* \Rightarrow \_^*$



which removes from a list elements of X.

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

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



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:

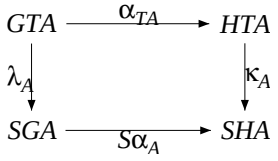
$$\begin{aligned}
k([],\,[]) &= [] \\
k(* :: \ell,\,[]) &= *_1 :: k(\ell,\,[]) \\
k(* :: \ell,\, a :: m) &= *_1 :: k(\ell,\, a :: m) \\
k([],\, * :: m) &= *_2 :: k(\mathrm{nil},\, m) \\
k(a :: \ell,\, * :: m) &= *_2 :: k(a :: \ell,\, m) \\
k(* :: \ell,\, * :: m) &= *_3 :: k(\ell,\, m) \\
k(a :: \ell,\,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

where we write [] as the empty list and $a :: \ell$ as the result of adding element $a \in A$ to list $\ell \in A^*$. An induction on the structure of Q shows that k is the inverse of h. □

3.2.4 Lifting cartesian natural transformations

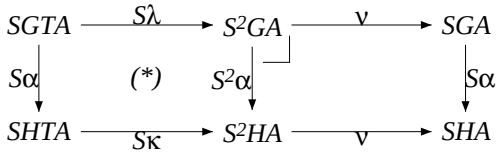
Suppose that G and H are cover-stable functors $(\mathbf{X}, \mathcal{X}) \to (\mathbf{Y}, \mathcal{Y})$ with cover-stable distributions λ and κ over cover-stable monads T and S respectively:

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



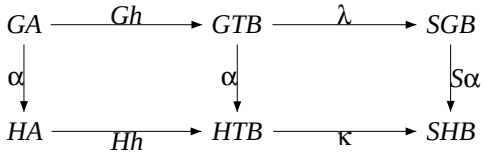
is a $\mathcal{Y}$-pullback for all A.

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:

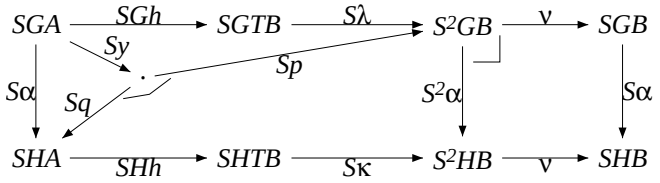


As ν is cartesian, (*) is a $\mathcal{Y}$-pullback and thus the required square is a $\mathcal{Y}$-pullback as S reflects $\mathcal{Y}$-pullbacks.

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



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

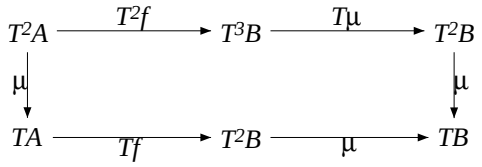


which implies $\tilde{\alpha}$ is $\mathcal{Y}_S$-cartesian. □

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 **Cov**: 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 $\epsilon_A; f = FUf; \epsilon_B$



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(**X**) be the subcategory of Dist(**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(**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(**X**) be the subcategory of Lift(**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(**X**) *is isomorphic to* CSLift(**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. □

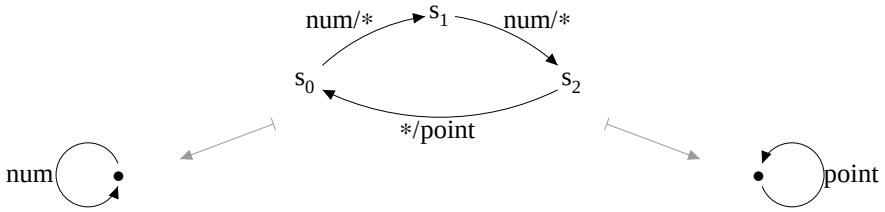
3.3 Example: Transitions systems

We now consider how the techniques of the previous section are used to reconstruct Abramsky's category ASProc of asynchronous processes modulo weak bisimulation [AGN94]. The construction can be seen as a functorial analogue to Milner's construction of asynchrony in process calculi [Mil83].

3.3.1 The delay monad

The delay monad on transition systems adds a new label (say $*$) and an idle transition on that label at every state:

The new label $*$ is interpreted as "do nothing". We can now write a process which, for example, translates a sequence of numbers into a sequence of points as a span

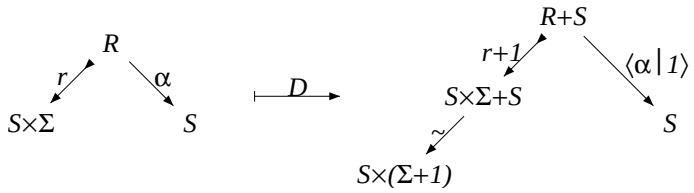


in the Kleisli category of the delay monad. Note that the labels chosen in the vertex indicate that each action has a visible effect at one interface only.

The delay monad exists in the category of transition systems upon any lextensive category $\mathbf{X}$.

Definition 3.3.1 *The delay monad D on* $\mathrm{Tran}(\mathbf{X})$ *has the following effect on a tran-*

sition system A:



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 a stable monad on* $\mathrm{Tran}(\mathbf{X})$.

Proof. It is easily shown that D is a functor and that η and μ are morphisms. The requirements for being a stable monad are then given by its construction componentwise in terms of the identity and exception monads. □

Consequently the Kleisli category $\mathrm{Tran}(\mathbf{X})_D$ has pullbacks. To see the effect of span composition here, let (P, f, g) and (Q, h, k) be composable spans in $\mathrm{Tran}(\mathbf{Set})_D$. Proposition 3.2.1 says that the apex of the composite span is the following transition system:

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

which is as one would expect for the composition of asynchronous processes (àla CCS [Mil89]): the components can either evolve together through visible communication, or can evolve independently through internal actions.

3.3.2 Weak bisimulation

In the interest of obtaining a cover system which ignores delay, consider the *path* construction on transition systems: for object A in Tran(**Set**) define $\mathrm{Path}(A) \stackrel{def}{=} (i \in S,\ \bigcup_{i<\omega} R^i)$, where

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

The transition system $\mathrm{Path}(A)$ has the same states as A, but its label set is Σ_A^* and its transitions correspond to all finite sequences of transitions in A.

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

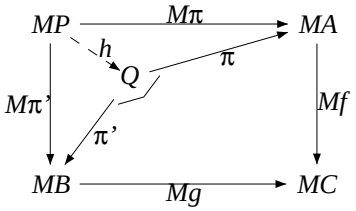
Proof. $\mathrm{Path}(A)$ is a deterministic transition system as all R_i are deterministic and involve distinct labels. The effect of Path on maps is given componentwise by the identity and list monads, and a simple induction on the structure of the labels shows this is well-defined.

To see that Path is stable, first recall the operations zip and unzip which translate between lists of pairs and pairs of lists:

$$\begin{aligned} \mathrm{zip}(a :: \ell_1,\ b :: \ell_2) &= (a,b) :: \mathrm{zip}(\ell_1, \ell_2) \\ \mathrm{zip}(a :: \ell_1,\ []) &= [] \\ \mathrm{zip}([],\ b :: \ell_2) &= [] \\ \mathrm{zip}([],\ []) &= [] \end{aligned}$$

$$\begin{aligned} \mathrm{unzip}([]) &= ([],[]) \\ \mathrm{unzip}((a,b) :: \ell) &= (a :: p_0(\mathrm{unzip}(\ell)),\ b :: p_1(\mathrm{unzip}(\ell))) \end{aligned}$$

Now let P be the pullback of f and g and consider the induced map h to the pullback of Path(f) and Path(g):



Define $h' : Q \to \mathrm{Path}(P)$ to have the identity effect on states and the following effect on labels: $(l, m) \mapsto \mathrm{zip}(l, m)$. 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 that zip;unzip is the identity on the subset $\{(\ell, m) \mid length(\ell) = length(m)\}$ of $A^*{\times}B^*$. □

The action $\theta : (\_ + 1)^* \Rightarrow \_^*$ of example 3.2.13 induces an action Path($D(\_)$) $\Rightarrow$ Path($\_$) for the path functor over the delay monad, which has the identity effect on states and the effect of θ on labels. For convenience we will refer to the induced action also as θ.

Example 3.3.4 $\theta : \mathrm{Path}(D(\_)) \Rightarrow \mathrm{Path}$ *is a stable action.*

Proof. An induction on the structure of the labels of Path(DA) shows that θ is a morphism of transition systems: suppose $(s, \ell, t) \in \mathrm{Path}(DA)$:

i) If $\ell = []$ then $s = t$, $\theta(\ell) = []$ and $(s, \theta(\ell), t) \in \mathrm{Path}(A)$ by definition.

ii) If $\ell = a :: m$ then there exists $u \in S_A$ such that $(s, a, u) \in DA$ and $(u, m, t) \in \mathrm{Path}(DA)$. Then either $a \in \Sigma_A$, so that $\theta(\ell) = a :: \theta(m)$, or $a = *$ in which case $u = s$ and $\theta(\ell) = \theta(m)$. In either case, $(u, \theta(m), t) \in \mathrm{Path}(A)$ by inductive hypothesis and so $(s, \theta(l), t) \in \mathrm{Path}(A)$ as required.

The fact that θ is a stable distribution is given by it's construction componentwise from the identity and the action $\theta : (\_ + 1)^* \Rightarrow \_^*$. □

This gives a stable functor $\mathrm{Path}_\theta : \mathrm{Tran}(\mathbf{Set})_D \to \mathrm{Tran}(\mathbf{Set})$ which is used to obtain a cover system for weak bisimulation. For $\exists^{\mathcal{E}}$ the local epimorphisms in $\mathrm{Tran}(\mathbf{Set})$, define

$$\exists^{\mathcal{E}}_\theta \stackrel{def}{=} \mathrm{Path}_\theta^{\dashv}(\exists^{\mathcal{E}})$$

To see how this cover system corresponds to weak bisimulation equivalence of asynchronous processes, 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 $p \stackrel{\ell}{\longrightarrow} p' \in \mathrm{Path}(P)$ implies that there exists q' and m such that $q \stackrel{m}{\longrightarrow} q' \in \mathrm{Path}(Q)$ 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 $\{(p,q) \in \mathcal{S} \mid f(p) = g(q)\}$, labels are $\{(x,y) \mid f(x) = g(y)\}$ and transitions are $\{(p,q) \stackrel{(x,y)}{\longrightarrow} (p',q') \mid p \stackrel{x}{\longrightarrow} p' \wedge q \stackrel{y}{\longrightarrow} q'\}$. Note that R is a subobject of the pullback of f and g which may ignore unreachable states. It is straightforward to show 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 [Mil83] rather than the description (given there as proposition 8.4) which has now become standard [Mil89].

For an alternative cover system, recall that the regular epimorphisms $\mathcal{E}$ in the category of trees are a cover system for trace equivalence. By post-composing the stable functor $\mathrm{Unfold} : \mathrm{Tran}(\mathbf{Set}) \to \mathrm{Tree}(\mathbf{Set})$ (the right adjoint) to the functor Path_θ, one obtains a cover system for weak trace equivalence:

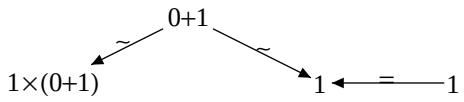
$$\mathcal{T}_W \stackrel{def}{=} (\mathrm{Path}_\theta ; \mathrm{Unfold})^{\dashv}(\mathcal{E})$$

3.3.3 Reformulating ASProc

We thus obtain a categories of asynchronous processes modulo both weak bisimulation and weak trace equivalence upon the Kleisli category of the delay monad:

$$\begin{aligned} \mathrm{ASProc} &\stackrel{def}{=} \mathrm{Proc}(\mathrm{Tran}(\mathbf{Set})_D, \exists^{\mathcal{E}}_{\theta}) \\ \mathrm{ASProc}_T &\stackrel{def}{=} \mathrm{Proc}(\mathrm{Tran}(\mathbf{Set})_D, \mathcal{T}_W) \end{aligned}$$

Concerning the structure of these process categories, note that adding delay to the initial transition system yields the final trannsition system:



This is true in any lextensive category $\mathbf{X}$, and thus by proposition 3.2.3:

Proposition 3.3.6 $\mathrm{Tran}(\mathbf{X})_D$ *has finite limits.*

Thus both ASProc and ASProc_T are compact-closed. Note, however, that the product in the Kleisli category is not the result of lifting the product from the category of transition systems: given transition systems A and B in $\mathrm{Tran}(\mathbf{Set})_D$, their product is given by the transition system

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

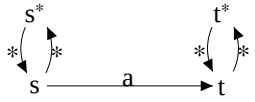
which allows transitions by both components simultaneously as well as independent transitions by either component.

Unfortunately, ASProc does not have biproducts. This can be seen by the fact that coproducts in $\mathrm{Tran}(\mathbf{Set})_D$ are not stable, which in turn is due to the fact that coproducts in $\mathbf{Set}_{\_+1}$ are not stable.

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

3.4 Example: Interleaved games

Consider the model category Game (example 2.5.2) 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 and their duplicates:



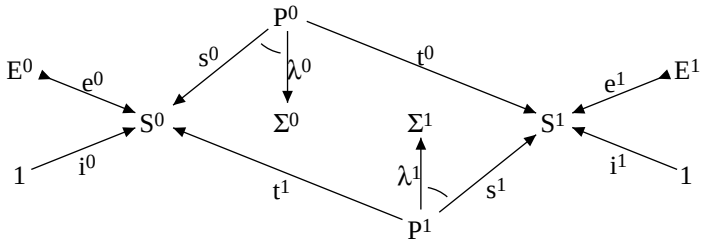
There is a problem, however, if the states introduced by delay have the same status as the original states: in essence, the monad

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

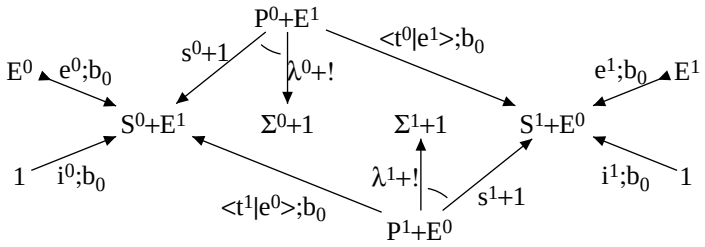
on $\mathbf{X}{\times}\mathbf{X}$ with unit (b_0, b_1) and multiplication (∇, ∇) is not stable. Consequently the suggested monad does not admit the construction of a process category.

The problem is overcome by distinguishing certain states of a game as being *external*. We thus modify the sketch of example 2.5.2 as follows:

Definition 3.4.1 $\mathrm{EGame}(\mathbf{X})$ *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.*

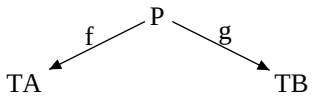
Proof. It is easily shown that D is a functor and that η and μ are morphisms. The requirements of being a stable monad are then given componentwise. □

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

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

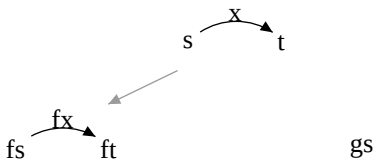
whose objects have only external states. Thus the non-external (or internal) states are explicitly used only in the implementation of processes.

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

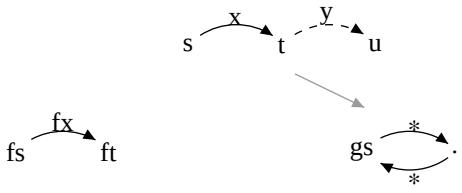


Suppose that s is a player state in P and that fs is external in A. Since Uf reflects external states, s must also be external. Furthermore, the fact that Uf uniquely reflects player transitions means not only that s admits in P all transitions permitted at fs in A, but that s admits no transition in 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.

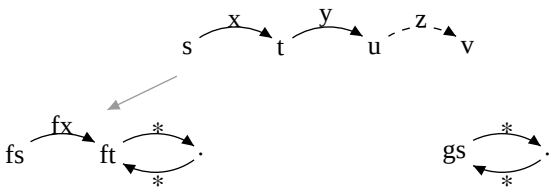


If the transition x is visible in B then t is an external opponent state at which P awaits input in B. Otherwise, x corresponds to idling in DB and (since g preserves external states) t is a non-external state which (since Ug uniquely reflects opponent transitions in DB) admits a unique transition y



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



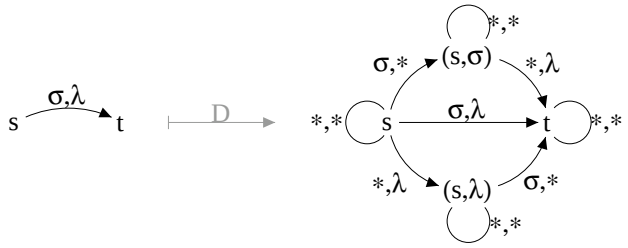
corresponding to the idle-back transition from fu in DA. The behavior of these processes should now be apparent: When in an external state, a process is reacting to the environment at the covered interface and thus cannot move until it receives input. Once a process initiates idling (as output at the non-covered interface), its evolution is determined to either perform internal actions "forever" or to eventually reach 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. I conjecture that the result of splitting these idempotents is Abramsky and Jagadeesan's category of games and strategies [AJ94], although not restricted to the winning strategies.

3.5 Example: Simultaneous games

Consider the games of example 2.5.1, in which both players move simultaneously. If each player has the ability to delay, then to maintain independence the delay monad must account for the possibility that one player specifies an action while the other

delays. In such cases the lone action must be encoded in the state space, since state change is not determined until both actions are given. The delay monad must then have the following effect on a transition:



Although the proof is incomplete, this operation appears to be a stable monad. It is necessarily more complicated than the delay monad on interleaved games, but shares certain similarities: one must add to the model category additional structure in the form of external states. Also, the cover systems for span construction are again lifted through the underlying functor of the monad and have the effect of restricting to processes which are deterministic and deadlock-free (although they may progress adinfinitum with no visible effect).

3.6 Summary

This chapter presents 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 the approach, we have used 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 [Mil83]. 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.

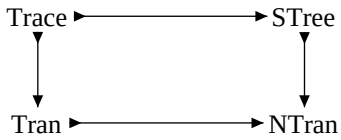
Note that although the delay monad exists in SProc, one does not obtain ASProc as a Kleisli construction upon SProc. This avenue is explored in Simon Gay's thesis [Gay95], where it is shown that the tensor (for parallel composition) in SProc does not have a natural distribution over the delay monad.

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 [SNW93], relationships are established between many standard models of concurrency: the linear-time models (e.g. traces and deterministic transition systems) are shown to be reflective subcategories of the corresponding branching-time models (e.g. synchronization trees and nondeterministic transition systems); the behavior models (e.g. traces and synchronization trees) form coreflective subcategories of the corresponding system models (e.g. transition systems). For the models of interleaved concurrency one has the following square of adjunctions,



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.

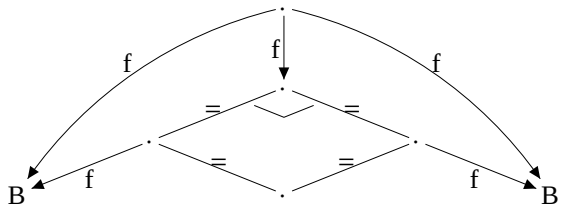
We develop a general notion of behavior object in a model category, and show that when a model category has enough behaviors it is sufficient to consider only the behavior objects when constructing a process category.

4.1 Moving between model categories

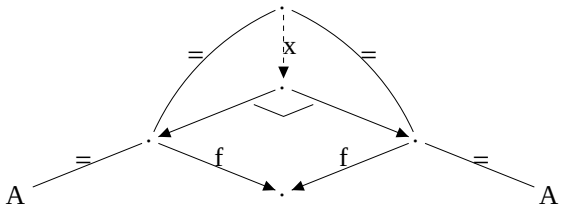
In some cases, adjunctions between model categories induce equivalences between the associated process categories. The key observation is that certain cover morphisms in a model category become isomorphisms in its process category.

Lemma 4.1.1 *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. Since $f \in \mathcal{X}$, the following diagram is a bisimulation demonstrating that $(f, 1); (1, f)$ is the identity on B.



Note in the diagram below that the induced morphism x is in $\mathcal{X}$ by left-factor closure: the diagram is thus a bisimulation equating $(1, f); (f, 1)$ with the identity on A.



So $(1, f)$ is an isomorphism with inverse $(f, 1)$. □

Consequently, a natural transformation in a model category 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 $\mathrm{Proc}(\alpha) : \mathrm{Proc}(F) \Rightarrow \mathrm{Proc}(G) : \mathrm{Proc}^{\mathcal{X}_0,\mathcal{X}_1}(\mathbf{X}, \mathcal{X}) \to \mathrm{Proc}^{\mathcal{Y}_0,\mathcal{Y}_1}(\mathbf{Y}, \mathcal{Y})$.

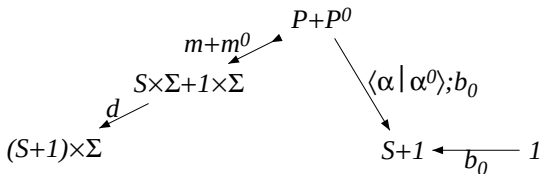
Corollary 4.1.2 *If $\mathcal{Y}'$ is a left-factor closed cover system such that $\mathcal{Y}' \subseteq \mathcal{Y}$ and α is componentwise in $\mathcal{Y}'$, then* $\mathrm{Proc}(\alpha)$ *is a natural isomorphism.*

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.

4.1.1 Separating initial states

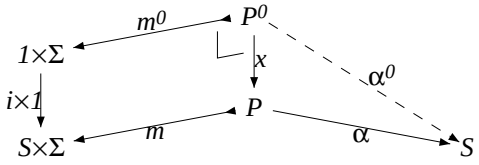
The model category of transition systems with separated initial state was introduced in section 2.4.2 to demonstrate a process category with biproducts. Here we show its process category coincides with that for ordinary deterministic transition systems.

For lextensive category $\mathbf{X}$, there is an adjunction $F \dashv G$ between the model categories $\mathrm{Tran}(\mathbf{X})$ and $\mathrm{STran}(\mathbf{X})$ which we will now define. 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:



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

thereby form an object GA of STran($\mathbf{X}$):

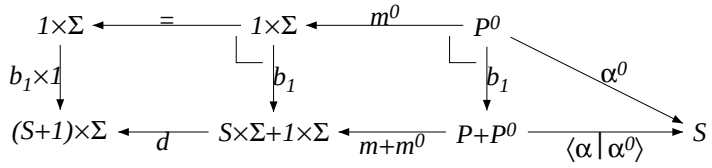


In STran($\mathbf{X}$), GFB differs from B in that it contains an unreachable copy of the initial state, and thus the unit η injects B into GFB. In Tran($\mathbf{X}$), FGA differs from A in that it has a new initial state which is bisimilar to the old (still reachable) initial state, so the counit ϵ_A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

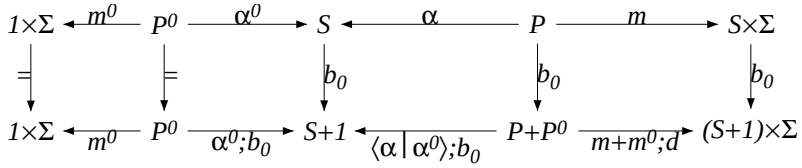
Proposition 4.1.3 Proc(STran($\mathbf{X}$), $\exists^{\mathcal{X}}$) *is equivalent to* Proc(Tran($\mathbf{X}$), $\exists^{\mathcal{X}}$) *whenever coproducts in* $\mathbf{X}$ *preserve* $\mathcal{X}$.

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

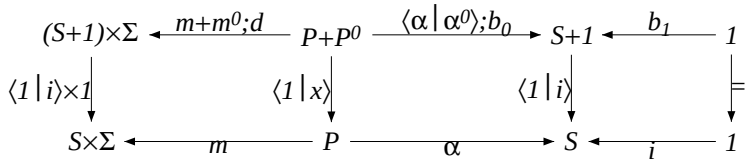


the unit η_B is given by:



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



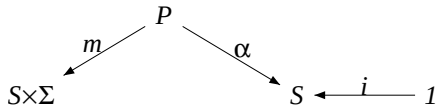
This morphism is in $\exists^{\mathcal{I}}$ also as coproducts are stable and ∇ is cartesian.

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

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.

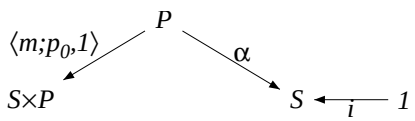
4.1.2 Nondeterministic transition systems

Consider the category of NTran(**X**) of nondeterministic transition systems given by the following sketch:



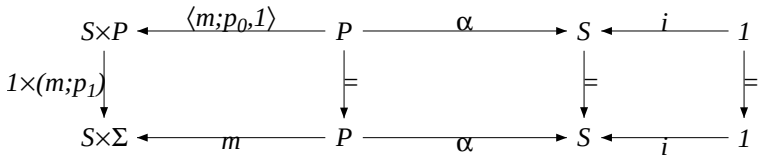
These transition systems differ from their deterministic counterparts only in that there may be many transitions with the same source state and label (viz. m is not monic).

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



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 : Tran(**X**) $\rightarrow$

NTran(**X**), the morphism



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:

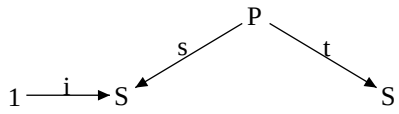
Proposition 4.1.4 Proc(NTran(**X**), $\exists^{\mathcal{X}}$) *is equivalent to* Proc(Tran(**X**), $\exists^{\mathcal{X}}$)

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 [SNW93]: the latter are Kleisli categories of appropriately defined *delay* monads upon Tran(**Set**) and (a slight modification of) NTran(**Set**), respectively.

4.1.3 Unlabelled transition systems

One obtains an equivalent process category beginning even with unlabelled transition systems. Define UTran(**X**) to be the category of models in **X** of the following sketch:



In this case there is an adjunction Tran(**X**) ⊣ UTran(**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 4.1.5 Proc(UTran(**X**), $\exists^{\mathcal{X}}$) *is equivalent to* Proc(Tran(**X**), $\exists^{\mathcal{X}}$).

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

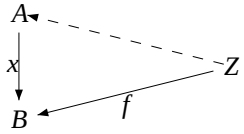
4.2 Behavior models

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.

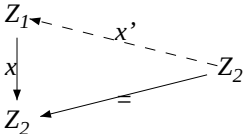
4.2.1 Abstract behaviors

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

Definition 4.2.1 *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:*



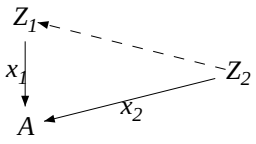
Intuitively, an $\mathcal{X}$-behavior cannot distinguish objects which are $\mathcal{X}$-equivalent. Note that any $\mathcal{X}$-morphism $x : Z_1 \to Z_2$ between $\mathcal{X}$-behaviors is an isomorphism: x has a left inverse x'



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

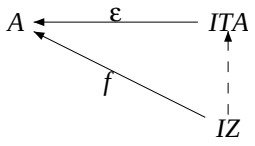
$\mathcal{X}$-behaviors for a given object are isomorphic:



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 4.2.2 *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:



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 4.2.3 *If $\mathcal{X} \subseteq \mathcal{Y}$ and $T(\mathcal{Y}) \subseteq \mathcal{Y}$ 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 2.3.12) $\mathcal{X}$-cartesian and thus $\mathcal{Y}$-cartesian. The coreflection is thus preserved by the process construction and (by proposition 4.1.2) becomes an equivalence. □

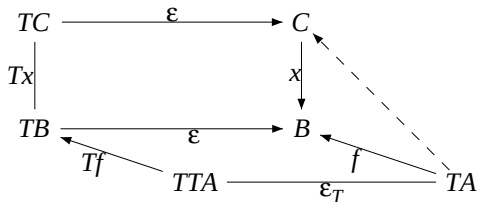
Any coreflective subcategory T of $\mathbf{C}$ determines a cover system for which T gives enough behaviors: the class $T^{-1}(\mathcal{I})$ (the morphisms of $\mathbf{C}$ taken by T to isomorphisms) is a left-factor closed cover system and, as T is an idempotent comonad [BW85], the counit ϵ belongs to $T^{-1}(\mathcal{I})$. In fact,

Proposition 4.2.4 *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 $f'; x = f$.



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.

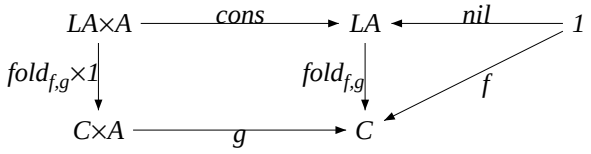
ii) If f is in $T^{-1}(\mathcal{I})$ then f is in $\mathcal{X}\downarrow$ since $\epsilon; f = Tf; \epsilon$ is in $\mathcal{X}$. Conversely, let x be witness that f is in $\mathcal{X}\downarrow$. Then Tx and $Tx; Tf$ are isomorphisms and thus Tf is an isomorphism.

□

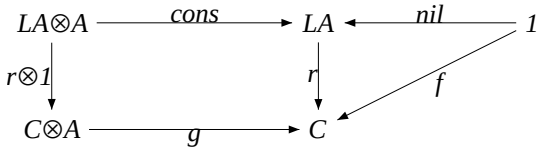
Behaviors of transition systems

A *locos* $\mathbf{C}$ is a lextensive category with list construction (see [Coc90]): i.e. there is an endofunctor L and natural transformations nil : $1 \to LA$ and 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 $\text{fold}_{f,g}$ such that

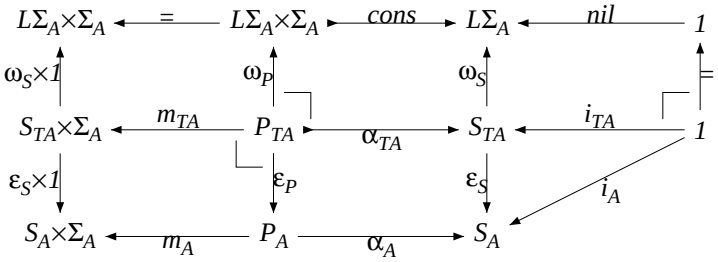


We say that a locos $\mathbf{C}$ has *span list construction* if given any $f : 1 \to C$ and $g : C \otimes A \to C$ in $\text{Span}(\mathbf{C})$ there exists r such that



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

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



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 [Hoa85] generated by A: i.e. S_{TA} is the nonempty and prefix-closed set of strings

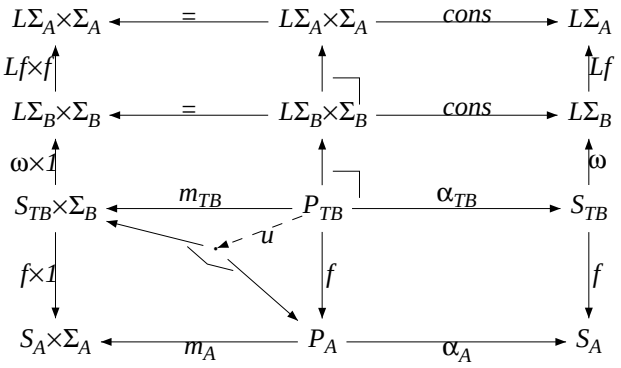
$$\{\, a_1 \ldots a_n \mid \exists s_1, \ldots s_n.\; i \xrightarrow{a_1} s_1 \longrightarrow \ldots \xrightarrow{a_n} s_n \,\},$$

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 (isosimilar) 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 [SNW93] 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,

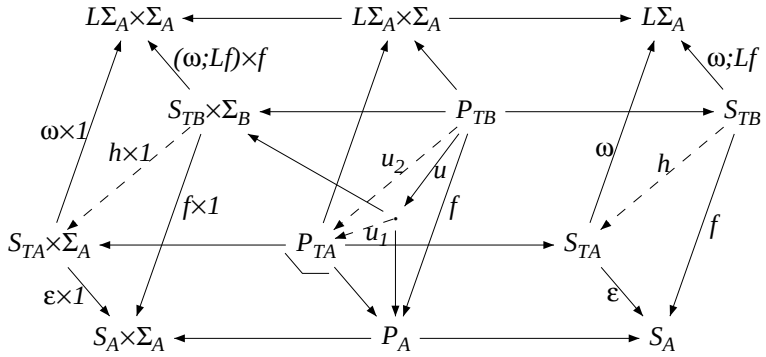
Proposition 4.2.5 Trace($\mathbf{C}$) *is a coreflective subcategory of* Tran($\mathbf{C}$), *and* STree($\mathbf{C}$) *is a coreflective subcategory of* NTran($\mathbf{C}$).

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



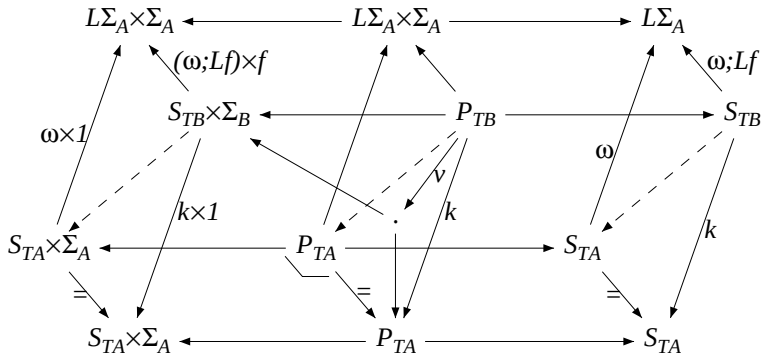
which is a 2-cell $u : \text{cons} \,;; (\omega_B; Lf, f) \Rightarrow (\omega_B \times 1 \,; Lf \times f, f \times f) \,;; (m_A, \alpha_A)$ in Span($\mathbf{C}$). By the universal property of span list construction, there exists $h : S_{TB} \to S_{TA}$ such

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 $\mathbf{C}$:

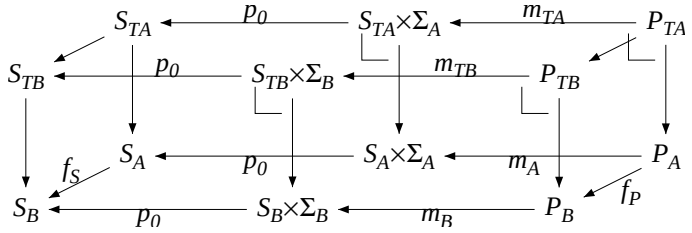


Consequently, k gives a 2-cell satisfying $u; k\times 1 ;; (m_A, \alpha_A) = \mathrm{cons} ;; k$ which forces $k = h$. □

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

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

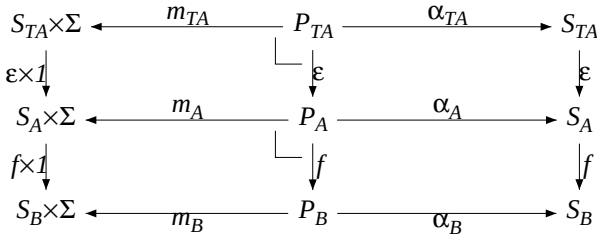


By lemma 2.2.4, if the bottom is an $\mathcal{X}$-pullback then the top will be an $\mathcal{X}$-pullback also. □

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 4.2.7 Tran($\mathbf{C}$) *and* NTran($\mathbf{C}$) *have* $\exists^{\mathcal{I}} \cap \delta^{-1}(\mathcal{I})$*-behaviors given by the categories* 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:

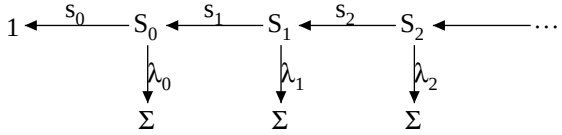


assuming for simplicity that $\Sigma_A = \Sigma_B = \Sigma$. □

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

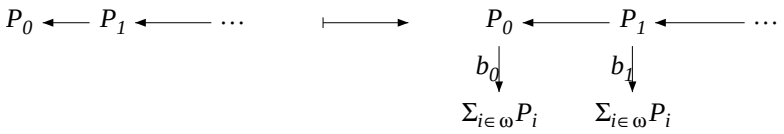
Proposition 4.2.8 *For the cover systems $\exists^{\mathcal{X}}$, the process categories constructed upon each of the following are equivalent:* Tran($\mathbf{C}$), NTran($\mathbf{C}$), Trace($\mathbf{C}$) *and* STree($\mathbf{C}$).

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 [Pav95]: i.e. STree($\mathbf{X}$) is the category of models in $\mathbf{X}$ of the following sketch:

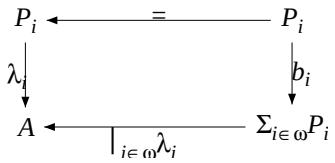


Similarly, the category of traces is obtained from the same sketch by requiring the labelling arrows λ_i to be monic. As in the category Tree($\mathbf{X}$) of trees in $\mathbf{X}$, the cover system $\exists^{\mathcal{X}}$ consists of the morphisms $f : A \to B$ which are $\mathcal{X}$-cartesian for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



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:



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

4.3 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 4.3.1 *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(**Set**)*, of deterministic transition systems;*
- NTran(**Set**)*, of nondeterministic transition systems;*
- UTran(**Set**)*, of unlabelled transition systems;*
- Trace(**Set**)*, of trace specifications;*
- STree(**Set**)*, of synchronization trees;*
- Tree(**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 [Coc95], 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.

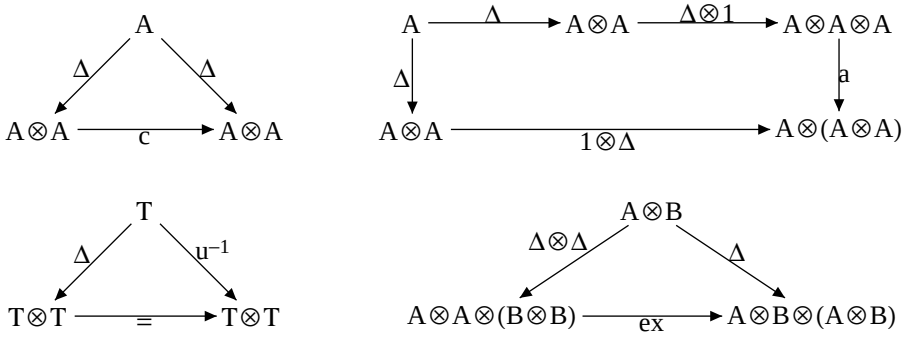
As the functional processes are the "maps" of SProc modulo trace equivalence, the category of circuits provides a natural setting for process implementation: one

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

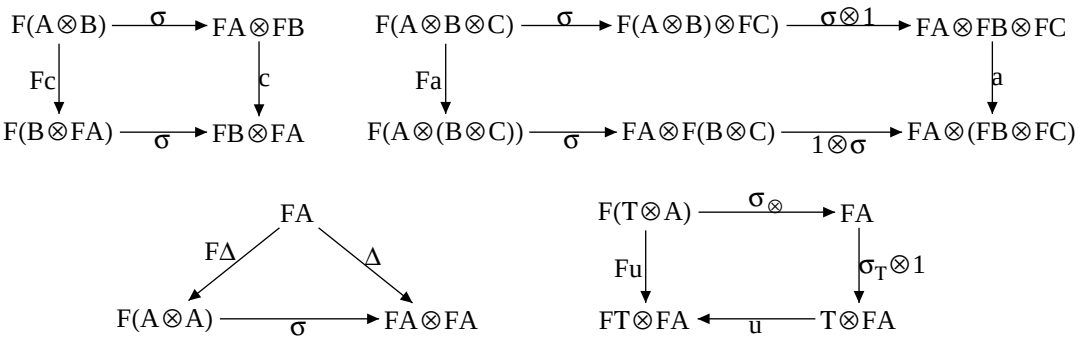
A *copy category* is a category with a symmetric tensor $(\otimes, \top)$ and a natural comultiplication Δ satisfying the following equations:



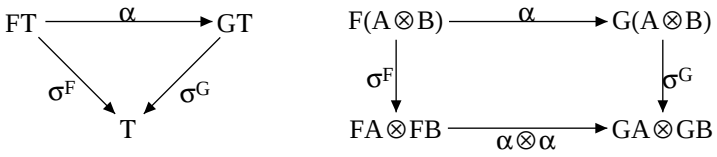
Note that a cartesian category, that is a category with finite products, is a copy category in which the tensor unit is the final object.

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

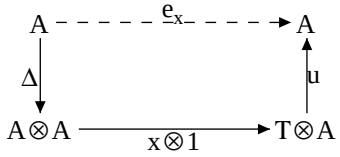


Similarly, a *copy transformation* is a natural transformation $\alpha : F \Rightarrow G$ between copy functors which is comonoidal: i.e.



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:



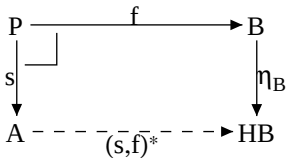
Here x is called a *copy proposition*, and e_x is called a *copy idempotent*. If e_x is the identity on A then x is the *counit* for A, and is denoted $!_A$. Counits are unique if they exist, and a copy category in which every object has a counit is said to be *strict*.

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 **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($\mathbf{C}$).

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



is a pullback. Mulry [Mul94b] 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 [Coc95].

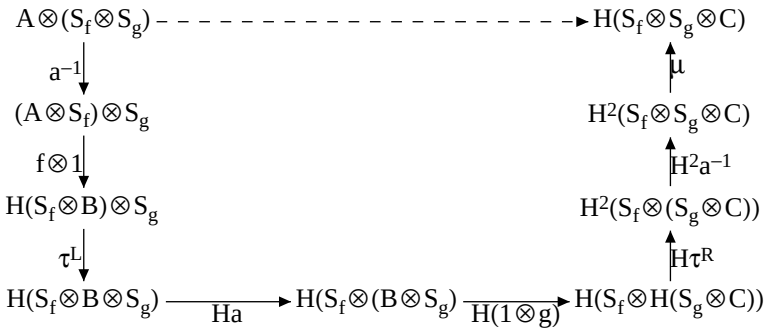
5.2 The category of circuits

In this section we review the circuit construction developed by Hensel and Spooner [HS96], and note the properties relevant to the subsequent sections. The circuit category described here differs from that of Katis, Sabadini and Walters [KSW94] 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.

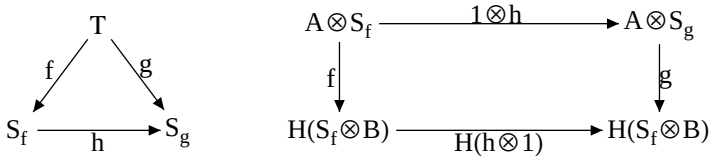
Let $\mathbf{X}$ be a copy category, and (H, η, μ) a copy monad on $\mathbf{X}$:

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*



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



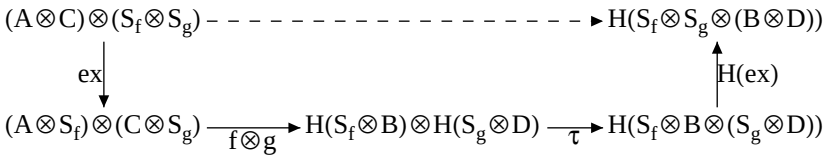
with identities and composition in **X**.

Taking **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.

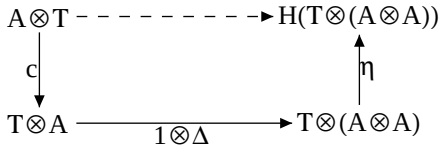
The presence of initial states has the effect that 2-cells are bisimulations, and bisimulation in each hom-category is just the equivalence relation induced by the 2-cells. In this way one obtains the category of partial circuits: [1]

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

The copy tensor $\otimes$ in **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:



The natural transformations associated with the copy structure of **X** also lift to the circuit category. For example, the copy transformation becomes the following circuit:



[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.

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.*

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 relates the categories of circuits described above with the subcategories of SProc presented in section 2.5.1. We begin by embedding the category of simply-typed circuits into the category DSProc of deterministic processes. Through splitting

propositional idempotents, this embedding lifts to an equivalence of the categories of safe circuits and of deterministic processes. Finally, taking the strict morphisms in this setting is shown to yield the category FSProc 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.

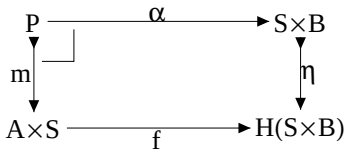
Let H be a separable partial map classifier on $\mathbf{X}$ with the property that the category of classified arrows has pullbacks:

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:

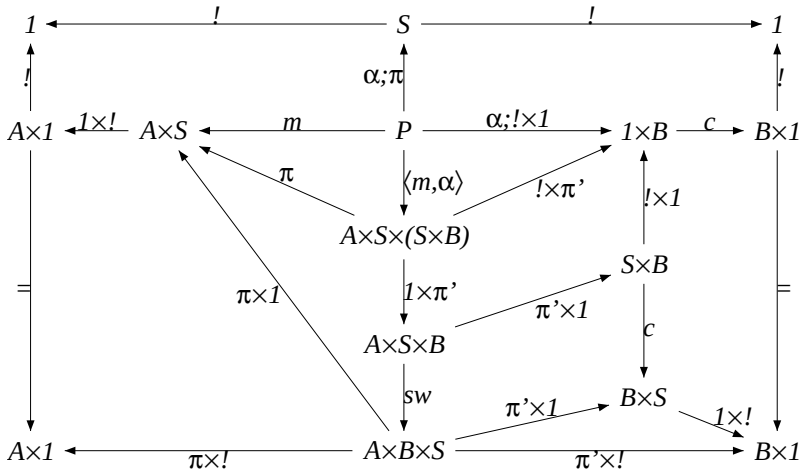
Definition 5.3.2 *Let* $\mathrm{DSProc}(\mathbf{X}, H)$ *be the process category upon* $\mathrm{Tran}(\mathbf{X}, H)$ *with left legs given by the local monics* $\exists^{\mathcal{M}_H}$ *and equivalence given by the local isomorphisms* $\exists^{\mathcal{I}}$.

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



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

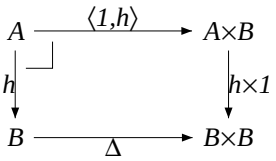
systems upon A and B respectively:



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.

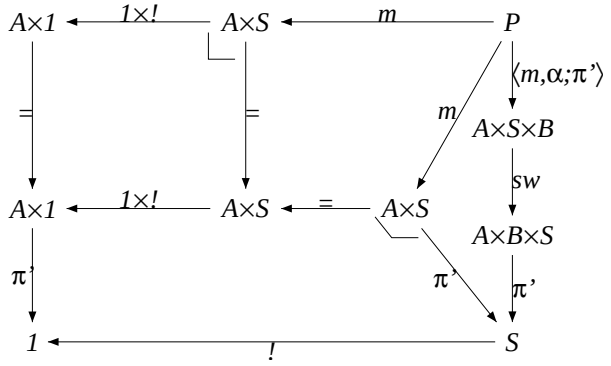
Proposition 5.3.3 *$F : \mathrm{Circ}(\mathbf{X}, H) \rightarrow \mathrm{DSProc}(\mathbf{X}, H)$ is a functor.*

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

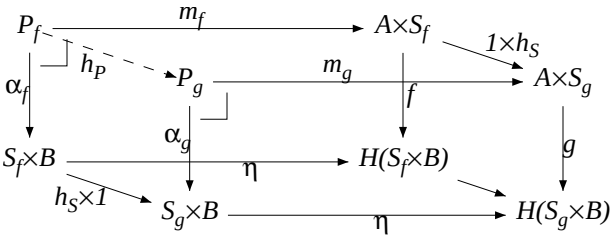


is a pullback and Δ is in $\mathcal{M}_H$. Thus $\langle m, \alpha; \pi'\rangle; sw = \langle 1, \alpha; \pi'\rangle; m{\times}1; sw$ is in $\mathcal{M}_H$.

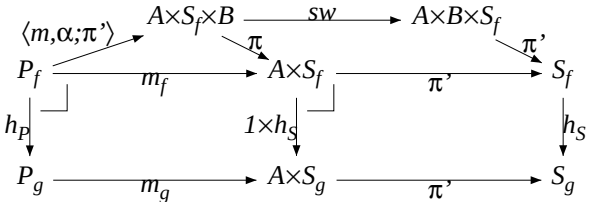
Then $F_L(f)$ is seen to belong to $\exists^{\mathcal{M}}$ as follows:



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



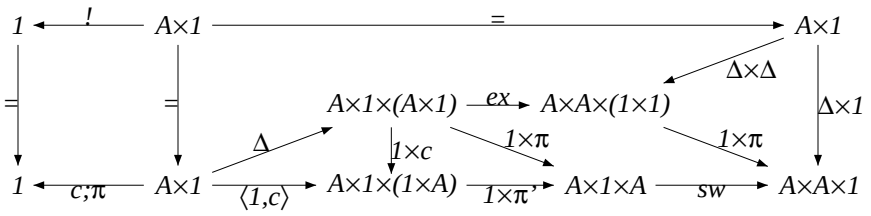
commutes, making h a morphism $F(f) \to F(g)$ in Tran. As η is cartesian the top face above is a pullback, and thus h is seen to belong to $\exists^{\mathcal{I}}$ as follows:



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.

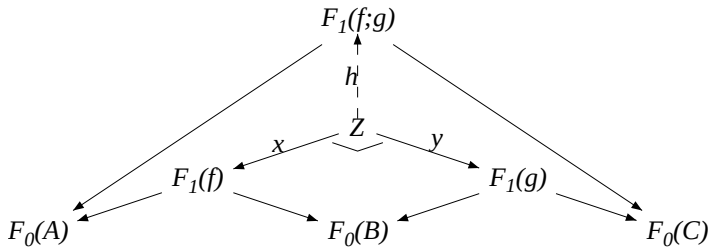
To show that F preserves identities, we suppose A in Circ and produce an $\exists^{\mathcal{I}}$ 2-cell $1_{F(A)} \to F(1_A)$ of DSProc. Recall that 1_A in Circ is given by $c;\eta : A\times 1 \to H(1\times A)$.

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

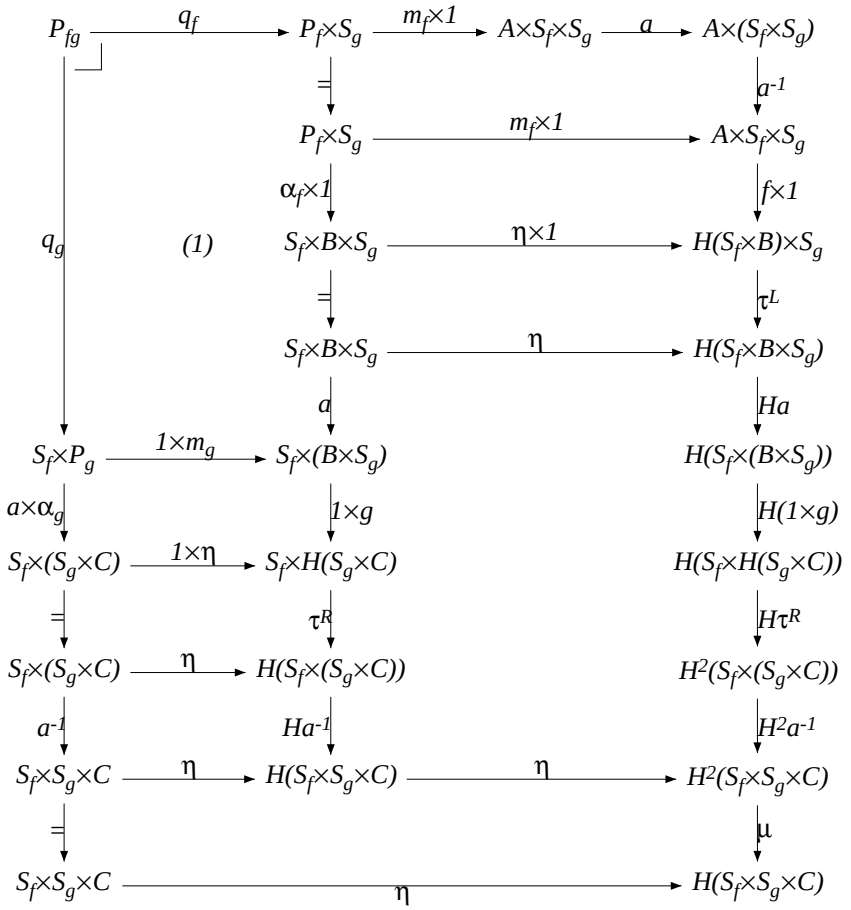


It belongs to $\exists^{\mathcal{I}}$ as the permission and state components are isomorphisms, and is easily checked to be a 2-cell.

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 $h : F(f); F(g) \to F(f; g)$.

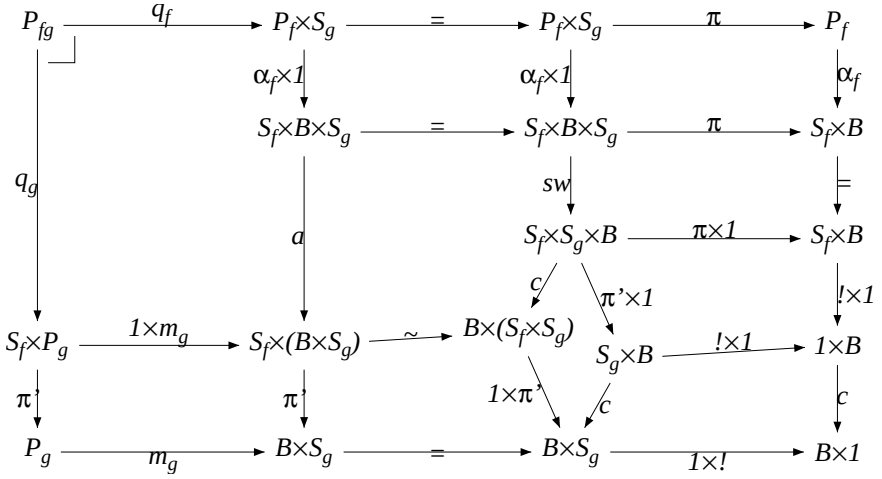


First we construct the partial map corresponding to $f;g$:

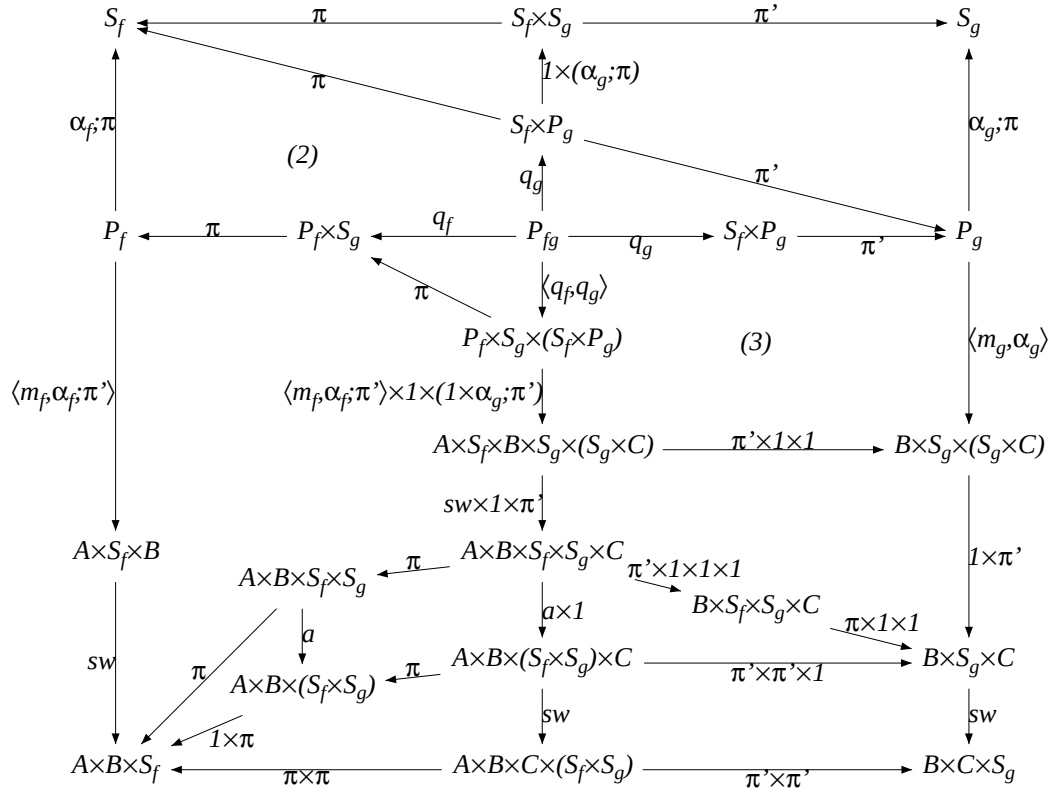


Now construct the pullback (Z, x, y) componentwise: the state component is $(S_f \times S_g, \pi, \pi')$; the label component is $(A \times B \times C, \pi, \pi' \times 1)$; and the permission component appears be-

low.

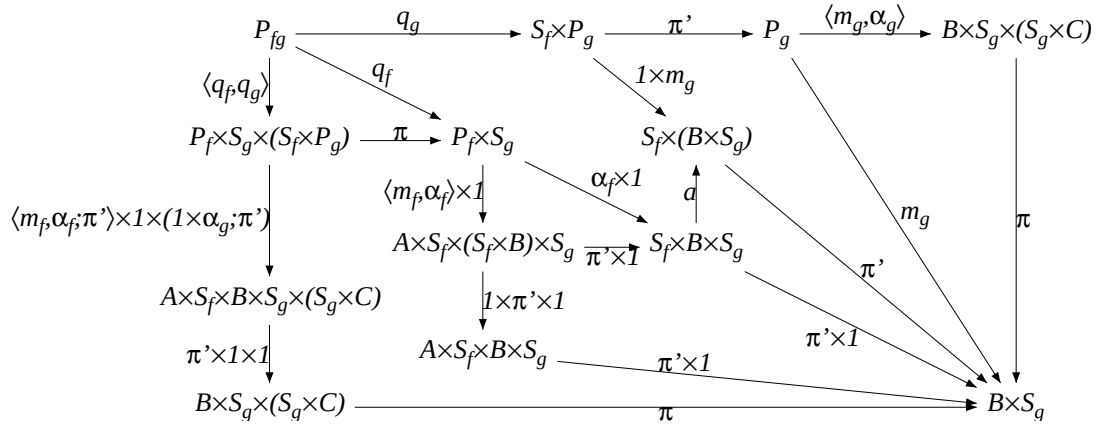


The transition system Z is given by the unique morphisms m_Z and α_Z such that $m_Z; x_\Sigma \times x_S = x_P; m_{Ff}$ and $m_z; y_\Sigma \times y_S = y_P; m_{Fg}$, and $\alpha_Z; x_S = x_P; \alpha_{Ff}$ and $\alpha_Z; y_S = y_P; \alpha_{Fg}$:

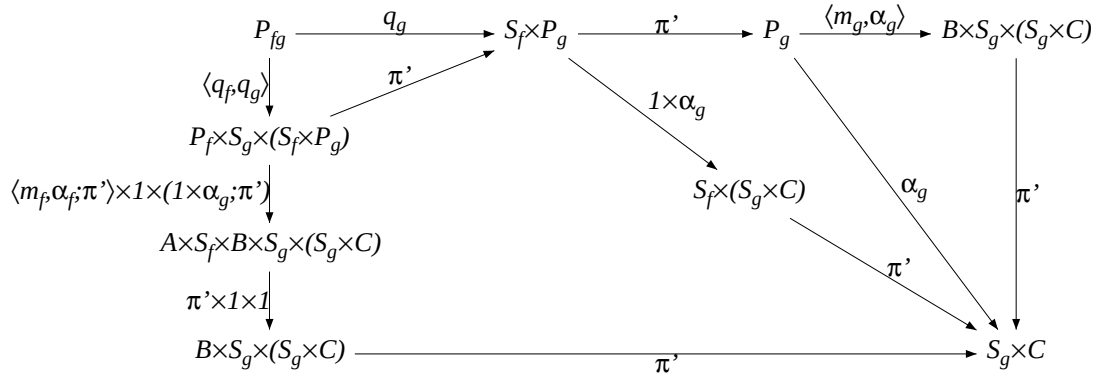


Note that (2) is (1);π. As a map into a product, we verify (3) componentwise: i.e.

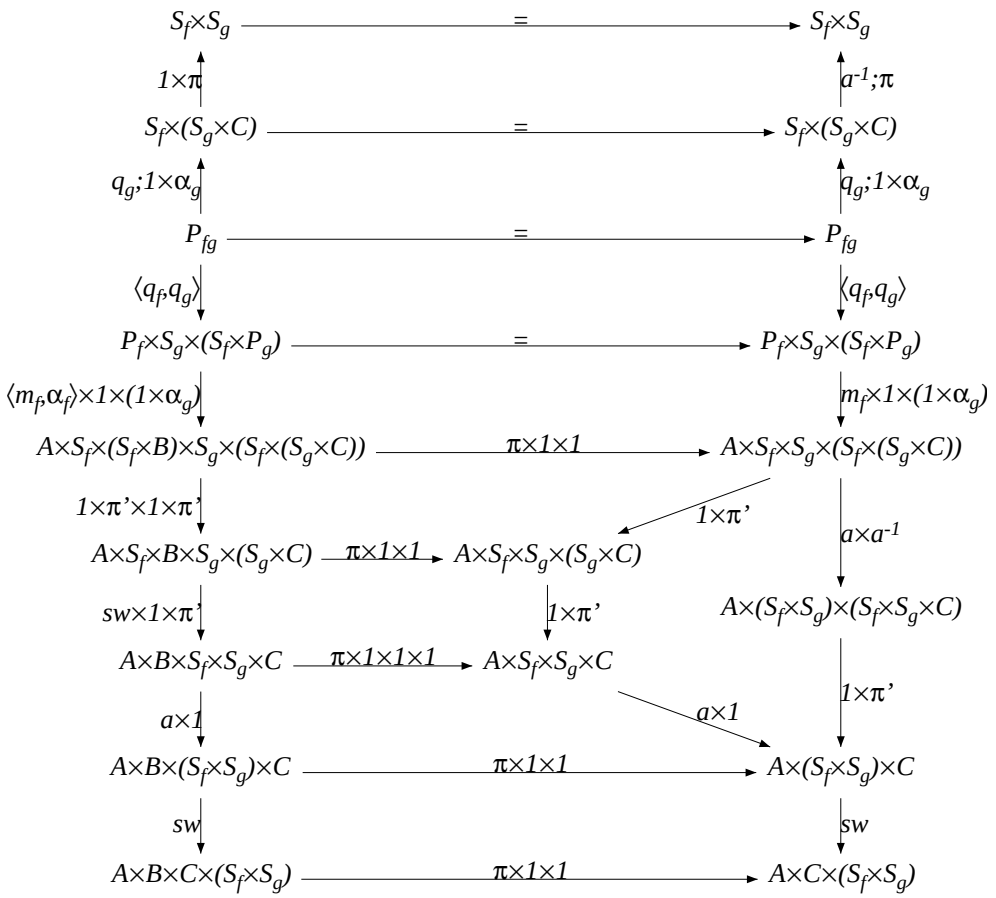
$(3);\pi$



and $(3);\pi'$ commute:



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



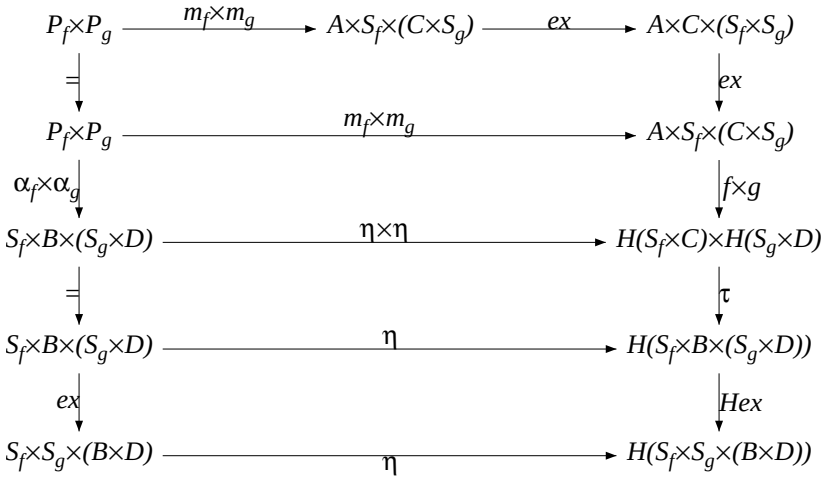
It is an $\exists^{\mathcal{I}}$-morphism as the state and permission components are isomorphisms, and is easily seen to be a 2-cell $F(f);F(g) \to F(f;g)$. □

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.

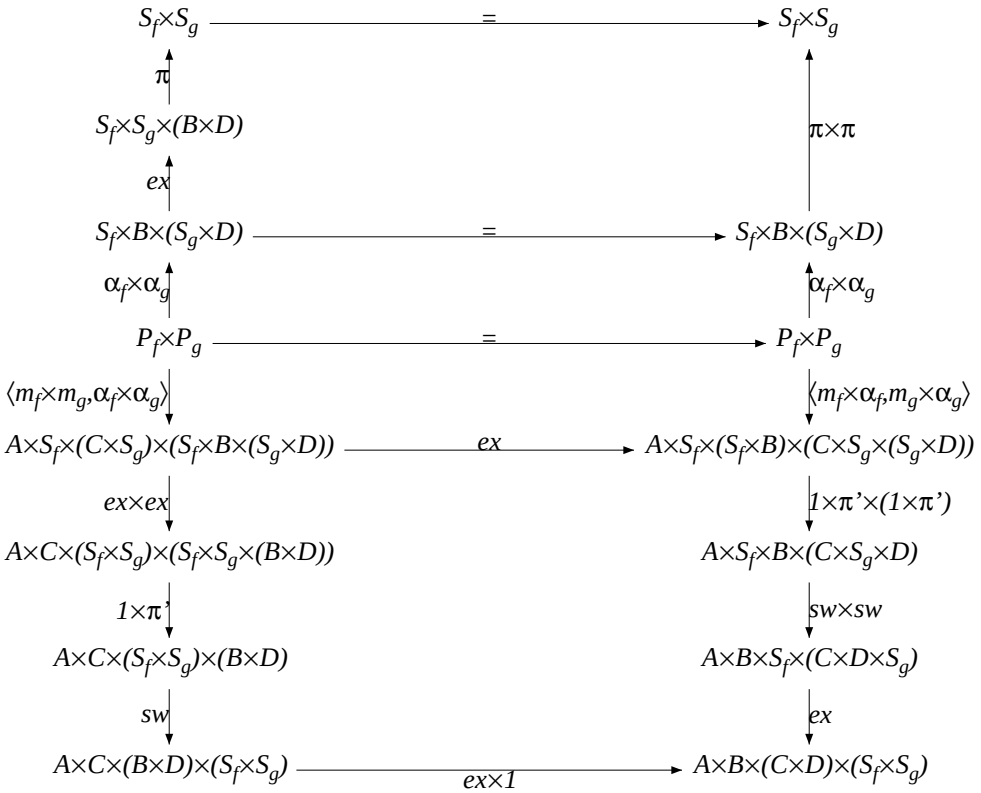
Proposition 5.3.4 *$F : \mathrm{Circ}(\mathbf{X}, H) \to \mathrm{DSProc}(\mathbf{X}, H)$ is a copy functor.*

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:



The required isomorphism in Tran is then:

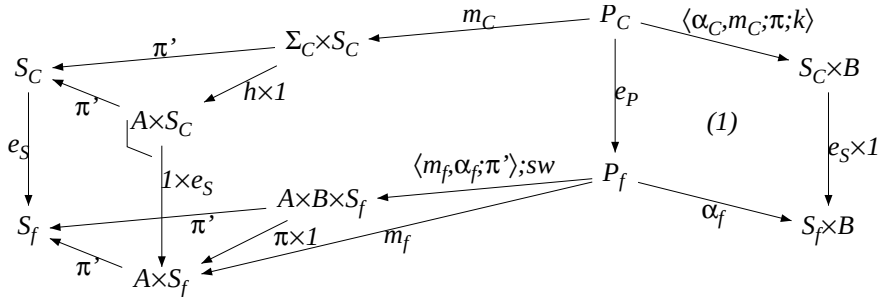


□

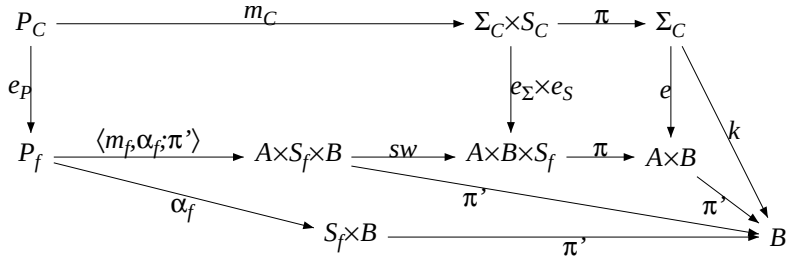
The fact that the encoding of circuits as processes is an embedding relies on the use of a separable partial map classifier.

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

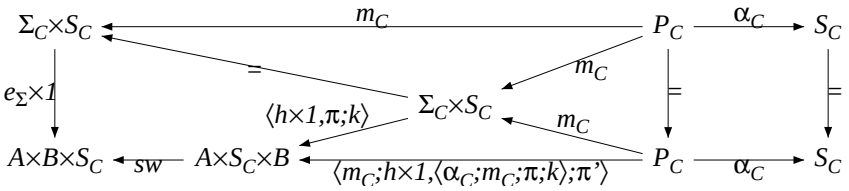
Proof. To show F is faithful, it is sufficient to show that for every $\exists^{\mathcal{I}}$ 2-cell $e : (h,k) \to F(f)$ of DSProc, there is a circuit f' and 2-cell $f' \to f$ with $F(f') = (h,k)$. Note that $e_\Sigma = \langle h_\Sigma, k_\Sigma \rangle : \Sigma_C \to A \times B$, and thus we have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



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

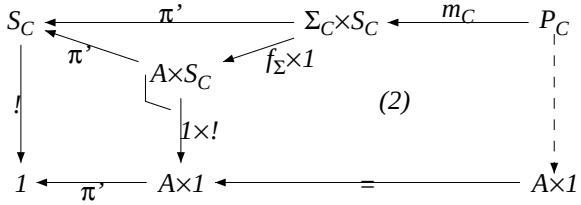


The fact that $e \in \exists^{\mathcal{I}}$ implies that $m_c; h \times 1$ is a pullback of m_f along e_P and thus an element of $\mathcal{M}$. Taking f' to be the associated circuit, we have $e_S : f' \to f$ in Circ. Finally, we have the following morphism $C \to F_1(f')$ of Tran

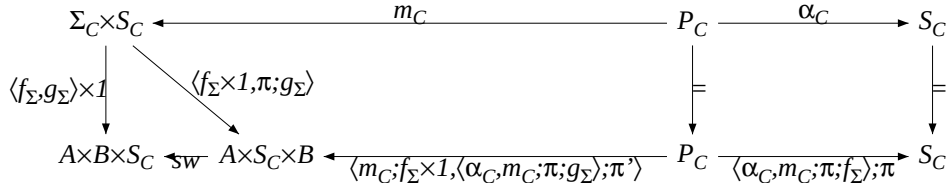


which is easily seen to be an $\exists^{\mathcal{I}}$ 2-cell $F(f') \to (h, k)$.

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 $\mathcal{M}$.



Let $h : A \to B$ 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:



□

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.

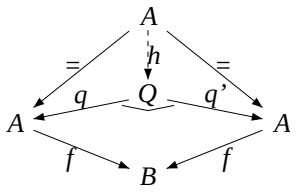
Lemma 5.3.6 *If $f : C \to A$ is in $\mathcal{C}_0 \cap \mathcal{C}_1$ and $1; f = 1; f$ is an $\mathcal{E}$-pullback, then $(f, f) : A \rightsquigarrow A$ is a split idempotent in $\mathrm{Proc}(\mathbf{X}, \mathcal{E}, \mathcal{C}_0, \mathcal{C}_1)$.*

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

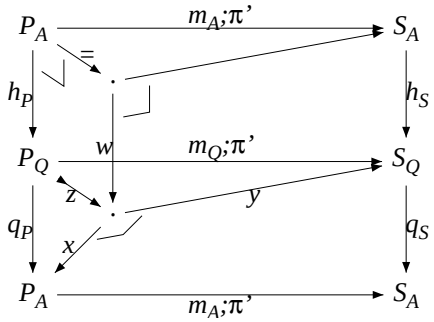
serve as $\mathcal{E}$ 2-cells $(f,f);(f,f) \to (f,f)$. In the category of transition systems, the *local* monics also have this property.

Lemma 5.3.7 *In* Tran(**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:



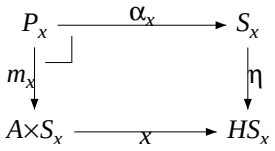
The fact that $h;q = 1$ induces w in the diagram below which makes $(m_a;\pi');h_S = w;y$ a pullback.



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

Consequently, any process (f,f) is a split idempotent in DSProc(**X**, H).

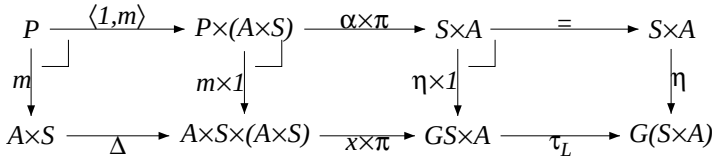
Returning to the embedding of circuits, we note that the copy propositions of Circ(**X**, H) correspond exactly to the objects of Tran(**X**, H):



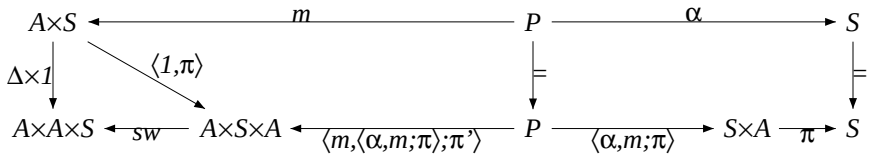
We define $G_0(x)$ to be the transition systems associated with $x : A \to 1$ of Circ, and $G_1(x) = (1_A, !_S) : G_0(x) \to F_0(A)$ the obvious morphism of Tran.

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:



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



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 $\exists^{\mathcal{I}}$. □

Thus $\mathrm{PSplit}(F)$ is a full and faithful functor $\mathrm{SafeCirc}(\mathbf{X}, H) \to \mathrm{DSProc}(\mathbf{X}, H)$. Furthermore, every object A of $\mathrm{DSProc}(\mathbf{X}, H)$ is isomorphic to $\mathrm{PSplit}(F)(e_x)$, 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 $\mathrm{PSplit}(F) : \mathrm{SafeCirc}(\mathbf{X}, H) \to \mathrm{DSProc}(\mathbf{X}, H)$ *is an equivalence of categories.*

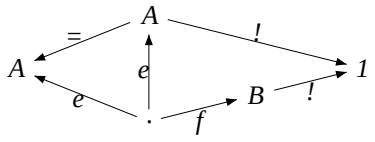
As DSProc is a strict copy category, we can ask what are the strict morphisms. Define

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

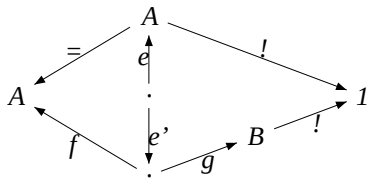
i.e. the subcategory of $\mathrm{DSProc}(\mathbf{X}, H)$ whose processes have their left leg a local isomorphism.

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

Proof. Note that the counit of an object A is given by the process $(1_A, !_A) : A \to 1$. If e is in $\exists^{\mathcal{I}}$ then any process $(e, f) : A \to B$ is strict, as 1 is final in Tran($\mathbf{X}, H$):



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



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 (via bisimulation) equations involving the circuit and its (proposed) type.

Chapter 6

Conclusion

This thesis has developed a general construction of process categories through a span construction upon a model category and a cover system.

I have shown that Abramsky's category SProc of synchronous processes arises as a process category upon all of the standard models of interleaved concurrency: from transition systems to trees. The fact that SProc is a process category upon trees gives directly the view, suggested in [AGN94], of SProc as a name-free approach to concurrency.

The fact that SProc is built upon trees also suggests a simple variation: in the category of trees, the cover system for trace equivalence is given by the regular epimorphisms. Thus one obtains the category SProc_T of synchronous processes modulo trace equivalence as the category of relations on trees. Since trees are essentially "sets in time", this suggests the proper interpretation of "relations in time" is actually SProc_T.

It was a general feeling that linear logic was the appropriate logic for interaction and process categories. Indeed, Abramsky's original presentations of interaction categories were through the type structure of linear logic. It is becoming clear, however, that (aside from games and strategies) few examples of interaction categories actu-

ally provide a full model of linear logic. For example to obtain storage in SProc one must throw away the fine distinctions provided by bisimulation and move to trace equivalence. In the asynchronous setting , ASProc, the situation is even worse as even the additives are lost. One must conclude that the appropriate levels of logic for describing these phenomena has still to be settled and will only become apparent as more examples surface.

One aspect of interaction categories which has not been given explicit attention here is that of specification structures. The idea there 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 which places stronger demands on the processes included in the category. One obtains a similar effect in the current setting by adding structure to the model category and using cover systems in the span construction to constrain the behavior of processes.

6.1 Future research

Some work remains to be done in the state-based formulation of asynchronous games of sections 3.4 and 3.3. Although I have provided a monads of delay which yield appropriate notions of asynchronous process, the equivalence on processes is still sensitive to delay. Left-factor closure of the span system forces the choice of cover system for equivalence, and thus an appropriate equivalence must be obtained by other means (e.g. a functor into ASProc).

For the interleaving games, the following steps remain in the reconstruction of the game-theoretic category of [AJ94]:

- A complete proof that the category of synchronous strategies is linearly distributive. This is made difficult by the extra accounting inherent in the state-based formulation of games.
- The fact that the obtained tensors preserve the span systems is enough to

transmit the linear distributive structure to the asynchronous process category.

- A functor into ASProc (given via the techniques of chapter 3) will yield an equivalence on processes which ignores delay, but the identity processes are still synchronous.
- Completing the idempotents which correspond to forcing asynchrony in the identity processes will then yield a *-autonomous category.

I believe that the game-theoretic categories provide the most promise as a foundation for concurrent programming. Consequently, the construction of circuit categories for games is an area which deserves further attention.

Bibliography

[Abr93] Samson Abramsky. Interaction categories (extended abstract). In *Theory and Formal Methods Workshop*. Springer Verlag, 1993.

[Acz88] Peter Aczel. Non-well-founded sets. *CLSI Lecture Notes*, 14, 1988.

[AGN94] 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.

[AJ94] Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543–574, 1994.

[AJM94] 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.

[Bar79] Michael Barr. **-autonomous categories*, volume 752 of *Lecture Notes in Mathematics*. Springer Verlag, 1979.

[Bar91] Michael Barr. *-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1:159–178, 1991.

[BBS88] D.B. Benson and O. Ben-Shachar. Bisimulation of automata. *Information and Computation*, 79:60–83, 1988.

[Bén67] J. Bénabou. Introduction to bicategories. *Lecture Notes in Mathematics*, 40:1–77, 1967.

[BPG94] 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.

[BW85] Michael Barr and Charles Wells. *Toposes, Triples and Theories*, volume 278 of *Grundlehren der mathematischen Wissenschaften*. Springer Verlag, 1985.

[BW90] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall, 1990.

[Cas85] I. Castellani. Bisimulation and abstraction homomorphisms. In *Proceedings of CAAP '85*. Springer Verlag, 1985. Lecture Notes in Computer Science.

[Coc90] J.R.B. Cockett. List-arithmetic open categories: Locoi. *Journal of Pure and Applied Algebra*, 66:1–29, 1990.

[Coc93] J.R.B. Cockett. Introduction to distributive categories. *Mathematical Structures in Computer Science*, 3:277–307, 1993.

[Coc95] J.R.B. Cockett. Copy categories. Submitted for publication, 1995.

[CS92] J.R.B. Cockett and R.A.G. Seely. Weakly distributive categories. *London Mathematical Society Lecture Notes*, 177:45–65, 1992.

[CS94] J.R.B. Cockett and David Spooner. Sproc categorically. *Lecture Notes in Computer Science*, 836:146–159, 1994. Proceedings of Concur '94.

[CS95] J.R.B. Cockett and David Spooner. Categories for synchrony and asynchrony. *Electronic Notes in Theoretical Computer Science*, 1:495–520, 1995.

[Gay95] Simon J. Gay. *Linear Types for Communicating Processes.* PhD thesis, London University, Imperial College, 1995.

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[HS96] 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.

[JM95] Andre Joyal and I. Moerdijk. *Algebraic Set Theory*, volume 220 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, 1995.

[JNW93] Andre Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation and open maps. In *Logic in Computer Science.* IEEE, 1993.

[Kat96] P. Katis. *Categories and bicategories of processes.* PhD thesis, The University of Sydney, 1996.

[Kel92] G.M. Kelly. On clubs and data-type constructors. In *Applications of Categories in Computer Science.* Cambridge University Press, 1992.

[KSW94] 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.

[Lin76] H. Lindner. A remark on mackey-functors. *Manuscripta Mathematica*, 18:273–278, 1976.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil91] Robin Milner. The polyadic π-calculus: A tutorial. LFCS Report Series ECS-LFCS-91-180, LFCS, Department of Computer Science, University of Edinburgh, 1991.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, 1990.

[Mul94a] Phillip S. Mulry. Lifting theorems for kleisli categories. In *Mathematical Foundations of Programming Semantics*, volume 802 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

[Mul94b] Phillip S. Mulry. Partial map classifiers and partial cartesian closed categories. *Theoretical Computer Science*, 136(1):109–123, 1994.

[Par90] Robert Paré. Simply connected limits. *Canadian Journal of Mathematics*, XLII(4):731–746, 1990.

[Pav95] Duško Pavlović. Convenient categories of processes and simulations 1: modulo strong bisimilarity. January 1995.

[Pra86] V.R. Pratt. Modeling concurrency with partial orders. *Int. J. of Parallel Programming*, 15(1):33–71, February 1986.

[Pra95] 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.

[See89] R.A.G. Seely. Linear logic, *-autonomous categories and cofree coalgebras. In J. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 371–382. American Mathematical Society, 1989.

[SNW93] 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.

[Tho91] Simon Thompson. *Type Theory and Functional Programming.* International Computer Science Series. Addison-Wesley, 1991.

[VB84] J. Van Bentham. Correspondence theory. In Gabbay and Guenther, editors, *Handbook of Philosophical Logic II.* Reidel, 1984.

[Wel94] Charles Wells. Sketches: Outline with references. Available by ftp from `ftp.cwru.edu` as `math/wells/sketch.ps`, February 1994.

[Win87] Glynn Winskel. *Event Structures*, volume 255 of *Lecture Notes in Computer Science.* Springer Verlag, 1987.