From Transitions to Executions*

Eleftherios Matsikoudis and Edward A. Lee

University of California, Berkeley

Abstract. Interleaving theories have traditionally failed to integrate a satisfactory treatment of the so-called "finite delay property". This is generally attributed to the expansion law of such theories, but in truth, the problem is rooted in the concept of labelled transition system. We introduce a new type of system, in which, instead of labelled transitions, we have, essentially, sequences of labelled transitions. We call systems of this type labelled execution systems. We use a coalgebraic representation to obtain, in a canonical way, a suitable concept of bisimilarity among such systems, study the conditions under which that concept agrees with the intuitive understanding of equivalence of branching structure that one has for these systems, and examine their relationship with labelled transition systems, precisely characterizing the difference in expressive power and branching complexity between the two kinds of systems.

1 Introduction

The process algebra literature is dominated by the concept of labelled transition system. And to some extent, this is understandable. For process algebra emerged from the marriage of Plotkin's structural operational semantics (see [32]) and Keller's named transition systems (see [21]) (see [28, chap. 12], [8], [33], [6]). This marriage was the work of Robin Milner, and is most clearly expounded in [28], but was already present in [24], where the so-called "expansion law" was stated for the first time.

The expansion law has been a constant source of controversy in the theory of concurrency. In the language of Milner's CCS (see [27], [28]), a typical equation asserted by the law is the following:

$$a.\mathbf{0} \mid b.\mathbf{0} = a.b.\mathbf{0} + b.a.\mathbf{0}. \tag{1}$$

* This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #1035672 (CPS: PTIDES), and #0931843 (ActionWebs)), the U. S. Army Research Office (ARO #W911NF-11-2-0038), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

Here, 'a' and 'b' stand for arbitrary actions, '$\mathbf{0}$' for the inactive agent, which is incapable of performing any action, '.' for sequential composition, '|' for parallel composition, and '+' for alternative composition. And the intended meaning of (1) is that the parallel execution of a and b is "equivalent", in some sense, to the indeterminate serialization of the two.

In order to justify the expansion law, and the blurring between causal dependence and temporal precedence resulting from it, Milner wrote the following in [24, p. 81]:

> We do not yet know how to frame a sufficiently general law without, in a sense, explicating parallelism in terms of non-determinism. More precisely, this means that we explicate a (parallel) composition by presenting all serializations - or interleavings - of its possible atomic actions. This has the disadvantage that we lose distinction between causally necessary sequence, and sequence which is fictitiously imposed upon causally independent actions; .... However, it may be justified to ignore it if we can accept the view that, in observing (communicating with) a composite system, we make our observations in a definite time sequence, thereby causing a sequencing of actions which, for the system itself, are causally independent.

Effectively, what he argued for was a dichotomy between causation and observation in the theory of concurrency. And what he proposed as an observational view to the theory was the interleaving of the atomic actions of the various agents inside a system as would be perceived by a single, sequential observer outside the system. But what he failed to admit was that the expansion law is in fact inconsistent with that view.

To understand the mismatch, consider the following equation derived from the expansion law, again in the language of CCS:

$$\mathbf{fix}(X = a.X) \mid \mathbf{fix}(X = b.X) = \mathbf{fix}(X = a.X + b.X). \tag{2}$$

Here, we use recursion expressions to define agents with infinite behaviour. Thus, $\mathbf{fix}(X = a.X)$ is an agent that forever iterates a, $\mathbf{fix}(X = b.X)$ one that forever iterates b, and $\mathbf{fix}(X = a.X + b.X)$ one that at each iteration, does either a or b, indeterminately choosing between the two. But whereas every infinite sequence over $\{a, b\}$ is a trace of a possible execution of $\mathbf{fix}(X = a.X + b.X)$, not every such sequence is consistent with what could be perceived by a sequential observer of $\mathbf{fix}(X = a.X) \mid \mathbf{fix}(X = b.X)$. Indeed, only those sequences that contain an infinite number of a's and an infinite number of b's are. For if $\mathbf{fix}(X = a.X)$ and $\mathbf{fix}(X = b.X)$ execute in parallel, each of them must eventually perform an infinite number of actions, and each of these actions must eventually be perceived by any sequential observer of $\mathbf{fix}(X = a.X) \mid \mathbf{fix}(X = b.X)$.

All this goes unnoticed in the finite case, because interleaving the executions of two finite agents is ultimately equivalent to indeterminately alternating between the two. But the expansion law blindly carried that equivalence over to

the infinite case. And this created confusion. Interleaving became synonymous with *bounded indeterminacy* (see [12, chap. 9]), and the observational view was robbed of its power to express properties like *fairness* (see [34]) or the *finite delay property* (see [20]) (e.g., see [30]).

Of course, it is not the expansion law per se that is to blame for this confusion. Interleaving is an operation on executions, not transitions. The problem is with the concept of labelled transition system: it is simply not up to the task.

The purpose of this work is to introduce a new kind of system, in which, instead of labelled transitions, one has, essentially, sequences of labelled transitions. We call systems of this kind *labelled execution systems*. Deferring the study of their potential uses as semantic models of behaviour, our goal here is to develop a thorough understanding of the so-called "branching structure" of such systems. And the theory of coalgebra offers a convenient and powerful analytical framework to do so.

The main contributions of this work are the following: (i) we define labelled execution systems, (ii) we represent these systems coalgebraically, and use that representation to obtain, in a canonical way, a suitable concept of bisimilarity among such systems, (iii) we study the conditions under which that concept agrees with the intuitive understanding of equivalence of branching structure that one has for these systems, and (iv) we work out their relationship with labelled transition systems, precisely characterizing the difference in expressive power and branching complexity between the two kinds of systems.

Throughout this work, we assume a basic level of familiarity with the theory of universal coalgebra, and in particular, with coalgebras over the category of all classes and all class functions between them. This is a "surprisingly coalgebra-friendly category" (see [5, p. 3]), which is one of the main reasons for choosing to work with it. And while it is possible to avoid such a super-large category, by only considering endofunctors that are bounded in some suitable sense, we think that classes provide for a cleaner, unobscured presentation of our ideas.

For lack of space, we omit all proofs; they can be found in [23].

2 Background

In this section, we recall a few of the most basic coalgebraic concepts that we use here, if for no other reason but to establish some common notation and terminology.

2.1 Coalgebras

Assume an endofunctor F on **Class**[1]

An *F-coalgebra* is an ordered pair $\langle C, \gamma \rangle$ of a class C and a class function $\gamma : C \to F(C)$.

Assume an F-coalgebra $\langle C, \gamma \rangle$.

We call C the *carrier* of $\langle C, \gamma \rangle$, and γ the *cooperation* of $\langle C, \gamma \rangle$.

We say that $\langle C, \gamma \rangle$ *is small* if and only if C is a set.

An F-coalgebra $\langle C, \gamma \rangle$ can be thought of as representing one or several rules for decomposing things from C, as determined by γ, into particular combinations of things from C, as encoded by F.

2.2 Homomorphisms

Assume F-coalgebras $\langle C_1, \gamma_1 \rangle$ and $\langle C_2, \gamma_2 \rangle$.

A *homomorphism* from $\langle C_1, \gamma_1 \rangle$ to $\langle C_2, \gamma_2 \rangle$ is a class function $h : C_1 \to C_2$ such that

$$F(h) \circ \gamma_1 = \gamma_2 \circ h.$$

A homomorphism from one F-coalgebra to another is a structure-preserving map carrying the decomposition patterns of the first coalgebra to those of the second, establishing a similarity of structure between its domain and range.

We write F-**Coalg** for the category whose objects are all the F-coalgebras, and arrows all the homomorphisms from one F-coalgebra to another.

2.3 Bisimulations

A *bisimulation* between $\langle C_1, \gamma_1 \rangle$ and $\langle C_2, \gamma_2 \rangle$ is a binary class relation $B : C_1 \leftrightarrow C_2$ such that there is a class function $\beta : \mathsf{graph}\, B \to F(\mathsf{graph}\, B)$ such that $\mathsf{dpr}\, B$ is a homomorphism from $\langle \mathsf{graph}\, B, \beta \rangle$ to $\langle C_1, \gamma_1 \rangle$, and $\mathsf{cpr}\, B$ is a homomorphism from $\langle \mathsf{graph}\, B, \beta \rangle$ to $\langle C_2, \gamma_2 \rangle$,[2]

We say that c_1 and c_2 are *bisimilar* among $\langle C_1, \gamma_1 \rangle$ and $\langle C_2, \gamma_2 \rangle$ if and only if there is a bisimulation B between $\langle C_1, \gamma_1 \rangle$ and $\langle C_2, \gamma_2 \rangle$ such that $c_1\ B\ c_2$.

---

[1] We write **Class** for the category whose objects are all the classes, and arrows all the class functions.

[2] For every binary class relation R, we write $\mathsf{dpr}\, R$ for a class function from $\mathsf{graph}\, R$ to $\mathsf{dom}\, R$ such that for any $\langle c_1, c_2 \rangle \in \mathsf{graph}\, R$, $(\mathsf{dpr}\, R)(\langle c_1, c_2 \rangle) = c_1$, and $\mathsf{cpr}\, R$ for a class function from $\mathsf{graph}\, R$ to $\mathsf{cod}\, R$ such that for any $\langle c_1, c_2 \rangle \in \mathsf{graph}\, R$, $(\mathsf{cpr}\, R)(\langle c_1, c_2 \rangle) = c_2$. We call $\mathsf{dpr}\, R$ the *domain projection map* of R, and $\mathsf{cpr}\, R$ the *codomain projection map* of R.

This coalgebraic notion of bisimulation was introduced in [4] as a generalization of the original concept of bisimulation between labelled transition systems (see Proposition 3 and 4). It is meant to capture equivalence of structure.

3 Labelled Transition Systems and Coalgebras

The concept of labelled transition system is of course the paradigmatic example of coalgebra. Indeed, the theory of coalgebra was largely inspired by the study of that concept (e.g., see [4]). Here, we formalize it in a slightly different way, namely using a binary relation rather than a ternary one, and go through its coalgebraic handling anew with the intent of drawing the reader's attention to the unity of formal treatment between this section and the next.

3.1 Labelled Transition Systems

Assume a non-empty set L of *labels*.

Definition 1. *An* L-labelled transition system *is an ordered pair* $\langle S, T\rangle$ *of a set* S *and a binary relation* $T : S \leftrightarrow L \times S$.

Assume an L-labelled transition system $\langle S, T\rangle$.

We write $s \xrightarrow{l}_T s'$ if and only if $s\ T\ \langle l, s'\rangle$.

We call any $s \in S$ a *state* of $\langle S, T\rangle$, and any $\langle s, \langle l, s'\rangle\rangle \in \mathsf{graph}\, T$ a *transition* of $\langle S, T\rangle$.

Labelled transition systems have been around at least since Moore's work on finite automata in [29], where they appeared in tabular as well as pictorial form. In their present form, they seem to have been introduced by Keller in [21], where they were called *named transition systems*. And although Keller used them to model parallel computation, it was apparently Milner who first saw labels as shared vehicles of interaction, and labelled transition systems as models of communicating behaviour, paving the way for [25] and the advent of process algebra.

Assume L-labelled transition systems $\langle S_1, T_1\rangle$ and $\langle S_2, T_2\rangle$.

Definition 2. *A* bisimulation between $\langle S_1, T_1\rangle$ and $\langle S_2, T_2\rangle$ *is a binary relation* $B : S_1 \leftrightarrow S_2$ *such that for any* s_1 *and* s_2 *such that* $s_1\ B\ s_2$*, the following are true:*

(a) if $s_1 \xrightarrow{l}_{T_1} s_1'$*, then there is* s_2' *such that* $s_2 \xrightarrow{l}_{T_2} s_2'$ *and* $s_1'\ B\ s_2'$*;*

(b) if $s_2 \xrightarrow{l}_{T_2} s_2'$*, then there is* s_1' *such that* $s_1 \xrightarrow{l}_{T_1} s_1'$ *and* $s_1'\ B\ s_2'$*.*

We say that s_1 and s_2 are *bisimilar* among $\langle S_1, T_1\rangle$ and $\langle S_2, T_2\rangle$ if and only if there is a bisimulation B between $\langle S_1, T_1\rangle$ and $\langle S_2, T_2\rangle$ such that $s_1\ B\ s_2$.

The idea of bisimilarity is that for any path branching out of either one of the two states, there is a path branching out of the other one, that carries the same labels in the same order, and goes through states that are again related to the corresponding states of the first path in the same way. This last piece of recursion is what separates bisimilarity from trace equivalence, making the former sensitive to the branching potential of each state.

The concept of bisimulation is due to David Park (see [31]), and is without doubt the most significant contribution of the theory of concurrency to the broader arena of computer science and mathematics at large.

3.2 Labelled Transition Coalgebras

Assume a binary relation $R : S_1 \leftrightarrow S_2$, and a function $f : S_1 \to \mathscr{P}\, S_2$.[3]

We write $\mathsf{fun}\, R$ for a function from S_1 to $\mathscr{P}\, S_2$ such that for any $s_1 \in S_1$,

$$(\mathsf{fun}\, R)(s_1) = \{s_2 \mid s_1\ R\ s_2\}.$$

We write $\mathsf{rel}\, f$ for a binary relation between S_1 and S_2 such that for any $s_1 \in S_1$ and any $s_2 \in S_2$,

$$s_1\ (\mathsf{rel}\, f)\ s_2 \iff s_2 \in f(s_1).$$

Proposition 1. *The following are true:*

(a) $\mathsf{rel}(\mathsf{fun}\, R) = R$;

(b) $\mathsf{fun}(\mathsf{rel}\, f) = f$.

Proposition 1 suggests an alternative, coalgebraic formalization of the concept of L-labelled transition system. The pertinent functor is of course the endofunctor $\mathsf{Pow} \circ (L \times \mathsf{Id})$ on **Class**, namely the composite of the power-set endofunctor Pow on **Class** with the left product endofunctor $L \times \mathsf{Id}$ on **Class**. An L-labelled transition system $\langle S, T\rangle$ can then be represented as a $(\mathsf{Pow} \circ (L \times \mathsf{Id}))$-coalgebra, namely as $\langle S, \mathsf{fun}\, T\rangle$, and conversely, a $(\mathsf{Pow} \circ (L \times \mathsf{Id}))$-coalgebra $\langle C, \tau\rangle$ as an L-labelled transition system, namely as $\langle C, \mathsf{rel}\, \tau\rangle$, with the caveat that C be a set.

Definition 3. *An* L-labelled transition coalgebra *is a* $(\mathsf{Pow} \circ (L \times \mathsf{Id}))$-*coalgebra.*

[3] For every set S, we write $\mathscr{P}\, S$ for the *power set* of S, namely the set of all subsets of S.

We write L-**LTC** for the category whose objects are all the L-labelled transition coalgebras, and arrows all the homomorphisms from one L-labelled transition coalgebra to another.

Formally, we will treat L-labelled transition systems and L-labelled transition coalgebras as distinct concepts. But informally, we shall think of an L-labelled transition coalgebra as an L-labelled transition system, no matter how large the carrier of the coalgebra is.

Assume an L-labelled transition coalgebra $\langle C, \tau \rangle$.

We write $c \xrightarrow{l}_{\tau} c'$ if and only if $\langle l, c' \rangle \in \tau(c)$.

Proposition 2. *The following are true:*

(a) $s \xrightarrow{l}_{T} s'$ *if and only if* $s \xrightarrow{l}_{\mathsf{fun}\, T} s'$*;*

(b) if $\langle C, \tau \rangle$ *is small, then* $c \xrightarrow{l}_{\tau} c'$ *if and only if* $c \xrightarrow{l}_{\mathsf{rel}\, \tau} c'$*.*

Assume L-labelled transition coalgebras $\langle C_1, \tau_1 \rangle$ and $\langle C_2, \tau_2 \rangle$.

Proposition 3. *B is a bisimulation between $\langle C_1, \tau_1 \rangle$ and $\langle C_2, \tau_2 \rangle$ if and only if B is a binary class relation between C_1 and C_2, and for any c_1 and c_2 such that $c_1\ B\ c_2$, the following are true:*

(a) if $c_1 \xrightarrow{l}_{\tau_1} c_1'$*, then there is* c_2' *such that* $c_2 \xrightarrow{l}_{\tau_2} c_2'$ *and* $c_1'\ B\ c_2'$*;*

(b) if $c_2 \xrightarrow{l}_{\tau_2} c_2'$*, then there is* c_1' *such that* $c_1 \xrightarrow{l}_{\tau_1} c_1'$ *and* $c_1'\ B\ c_2'$*.*

Proposition 4. *B is a bisimulation between $\langle S_1, T_1 \rangle$ and $\langle S_2, T_2 \rangle$ if and only if B is a bisimulation between the L-labelled transition coalgebras $\langle S_1, \mathsf{fun}\, T_1 \rangle$ and $\langle S_2, \mathsf{fun}\, T_2 \rangle$.*

4 Labelled Execution Systems and Coalgebras

This section contains the main body of this work, where we define and study our newly proposed systems. For the most part, we treat these systems in the guise of coalgebras. But for the benefit of the non-coalgebraist, as well as the more application-oriented reader, we treat systems and coalgebras as formally distinct, and translate our results back into the relational language of systems.

4.1 Labelled Execution Systems

Definition 4. *An* L-labelled execution system *is an ordered pair $\langle S, E \rangle$ of a set S and a binary relation $E : S \leftrightarrow \mathscr{S}(L \times S)$.*[4]

[4] For every set S, we write $\mathscr{S}\, S$ for the *sequence set* of S, namely the set of all finite and infinite sequences over S.

Assume an L-labelled execution system $\langle S, E\rangle$.

We write $s \rhd_E e$ if and only if $s\ E\ e$.

We call any $s \in S$ a *state* of $\langle S, E\rangle$, and any $\langle s, e\rangle \in \mathsf{graph}\, E$ an *execution* of $\langle S, E\rangle$.

Notice that an execution is an ordered pair of a state and a sequence of ordered pairs of labels and states, instead of a single odd-length alternating sequence of states and labels, what might have seemed a more natural option. And while we do think that there is a certain clarity in distinguishing the starting state of an execution from any subsequent step, this was mainly a choice of mathematical convenience. Its merit will soon become apparent.

4.2 Labelled Execution Coalgebras

As our choice of formalization should have made obvious, the concept of labelled execution system is a direct generalization of that of labelled transition system. The idea of a single step from one state to another is replaced by that of an "admissible" path through the system over which a sequence of steps can be taken. The result is a more elaborate notion of branching structure. And if we are to associate this notion with behaviour of some kind, we need to understand what constitutes similarity and dissimilarity of it. In other words, we need a concept of branching equivalence suited to labelled execution systems. What should that concept be?

In [37], Rutten and Turi propose a simple approach to this type of problem: all we have to do is find a suitable endofunctor to represent our systems coalgebraically. We can then use that endofunctor to instantiate the "parametric" concept of coalgebraic bisimulation and obtain not only the equivalence concept that we seek, but a model too that is *fully abstract* with respect to that concept.[5] This is straightforward here.

We begin by composing the sequence-set endofunctor Seq on **Class** with the left product endofunctor $L \times \mathsf{Id}$ on **Class** to obtain the endofunctor $\mathsf{Seq} \circ (L \times \mathsf{Id})$ on **Class**, which assigns to every class C the class

$$\mathsf{Seq}(L \times C) = \{s \mid \text{there is } S \text{ such that } S \subseteq L \times C \text{ and } s \in \mathscr{S}\, S\},$$

and to every class function $f : C_1 \to C_2$ a class function $\mathsf{Seq}(L \times f)$ from $\mathsf{Seq}(L \times C_1)$ to $\mathsf{Seq}(L \times C_2)$ such that for every $s \in \mathsf{Seq}(L \times C_1)$,

$$(\mathsf{Seq}(L \times f))(s) = \begin{cases} \langle\ \rangle & \text{if } s = \langle\ \rangle; \\ \langle\langle \mathsf{first\ head}\ s, f(\mathsf{sec\ head}\ s)\rangle\rangle \cdot (\mathsf{Seq}(L \times f))(\mathsf{tail}\ s) & \text{otherwise.} \end{cases}$$

[5] The tacit assumption here is that the instantiated concept of bisimulation does indeed induce an equivalence concept, which is not always true.

Despite the seeming circularity, the action of $\mathsf{Seq} \circ (L \times \mathsf{Id})$ on class functions is well defined: it is an instance of a definition by corecursion (see [7]).

We can now compose Pow with the endofunctor $\mathsf{Seq} \circ (L \times \mathsf{Id})$ to obtain the endofunctor $\mathsf{Pow} \circ \mathsf{Seq} \circ (L \times \mathsf{Id})$ on **Class**, which assigns to every class C the class

$$\mathsf{Pow}\,\mathsf{Seq}(L \times C) = \{S \mid S \text{ is a set, and } S \subseteq \mathsf{Seq}(L \times C)\},$$

and to every class function $f : C_1 \to C_2$ a class function $\mathsf{Pow}\,\mathsf{Seq}(L \times f)$ from $\mathsf{Pow}\,\mathsf{Seq}(L \times C_1)$ to $\mathsf{Pow}\,\mathsf{Seq}(L \times C_2)$ such that for every $S \in \mathsf{Pow}\,\mathsf{Seq}(L \times C_1)$,

$$(\mathsf{Pow}\,\mathsf{Seq}(L \times f))(S) = \{(\mathsf{Seq}(L \times f))(s) \mid s \in S\}.$$

Just as with labelled transition systems, we can use Proposition 1 to obtain our coalgebraic representation.

Definition 5. *An* L-labelled execution coalgebra *is a* $(\mathsf{Pow} \circ \mathsf{Seq} \circ (L \times \mathsf{Id}))$-*coalgebra.*

We write L-**LEC** for the category whose objects are all the L-labelled execution coalgebras, and arrows all the homomorphisms from one L-labelled execution coalgebra to another.

Assume an L-labelled execution coalgebra $\langle C, \varepsilon \rangle$.

We write $c \rhd_\varepsilon e$ if and only if $e \in \varepsilon(c)$.

Proposition 5. *The following are true:*

(a) $s \rhd_T e$ *if and only if* $s \rhd_{\mathsf{fun}\,T} e$*;*

(b) if $\langle C, \varepsilon \rangle$ *is small, then* $c \rhd_\varepsilon e$ *if and only if* $c \rhd_{\mathsf{rel}\,\varepsilon} e$*.*

At this stage, we could already use Proposition 7 below as our definition of bisimulation between labelled execution systems. But we prefer a different, more operational one that will help us develop some insight into the concept.

Assume a binary class relation $R : C_1 \leftrightarrow C_2$.

We write $\mathsf{Seq}(L \times R)$ for a binary class relation between $\mathsf{Seq}(L \times C_1)$ and $\mathsf{Seq}(L \times C_2)$ such that for every $e_1 \in \mathsf{Seq}(L \times C_1)$ and every $e_2 \in \mathsf{Seq}(L \times C_2)$,

$$\begin{aligned} e_1\ \mathsf{Seq}(L \times R)\ e_2 \iff\ & \text{there is } e \in \mathsf{Seq}(L \times \mathsf{graph}\,R) \\ & \text{such that } e_1 = \mathsf{Seq}(L \times \mathsf{dpr}\,R)(e) \\ & \text{and } e_2 = \mathsf{Seq}(L \times \mathsf{cpr}\,R)(e). \end{aligned}$$

Notice that $\mathsf{Seq}(L \times R)$ is a simple lift of R to pairs of sequences over $L \times \mathsf{dom}\,R$ and $L \times \mathsf{cod}\,R$. Our choice of notation may be justified by the fact that

$$\begin{aligned} \mathsf{Seq}(L \times R) &= \mathsf{Seq}(L \times ((\mathsf{dpr}\,R)^{-1}\,;\mathsf{cpr}\,R)) \\ &= \mathsf{Seq}(L \times (\mathsf{dpr}\,R)^{-1})\,;\mathsf{Seq}(L \times \mathsf{cpr}\,R). \end{aligned}$$

Assume L-labelled execution coalgebras $\langle C_1, \varepsilon_1 \rangle$ and $\langle C_2, \varepsilon_2 \rangle$.

Proposition 6. *B is a bisimulation between $\langle C_1, \varepsilon_1 \rangle$ and $\langle C_2, \varepsilon_2 \rangle$ if and only if B is a binary class relation between $\langle C_1, \varepsilon_1 \rangle$ and $\langle C_2, \varepsilon_2 \rangle$, and for any c_1 and c_2 such that $c_1\ B\ c_2$, the following are true:*

(a) if $c_1 \rhd_{\varepsilon_1} e_1$, then there is e_2 such that $c_2 \rhd_{\varepsilon_2} e_2$ and $e_1\ \mathsf{Seq}(L \times B)\ e_2$;

(b) if $c_2 \rhd_{\varepsilon_2} e_2$, then there is e_1 such that $c_1 \rhd_{\varepsilon_1} e_1$ and $e_1\ \mathsf{Seq}(L \times B)\ e_2$.

Proposition 6 is similar to Proposition 3. The difference is that the local check of correspondence of transitions has been replaced by a non-local test of agreement along entire executions.

Assume L-labelled execution systems $\langle S_1, E_1 \rangle$ and $\langle S_2, E_2 \rangle$.

Definition 6. *A* bisimulation between $\langle S_1, E_1 \rangle$ and $\langle S_2, E_2 \rangle$ *is a binary relation $B : S_1 \leftrightarrow S_2$ such that for any s_1 and s_2 such that $s_1\ B\ s_2$, the following are true:*

(a) if $s_1 \rhd_{E_1} e_1$, then there is e_2 such that $s_2 \rhd_{E_2} e_2$ and $e_1\ \mathsf{Seq}(L \times B)\ e_2$;

(b) if $s_2 \rhd_{E_2} e_2$, then there is e_1 such that $s_1 \rhd_{E_1} e_1$ and $e_1\ \mathsf{Seq}(L \times B)\ e_2$.

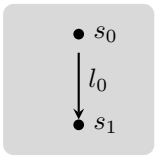
We say that s_1 *and* s_2 *are bisimilar among* $\langle S_1, E_1 \rangle$ *and* $\langle S_2, E_2 \rangle$ if and only if there is a bisimulation B between $\langle S_1, E_1 \rangle$ and $\langle S_2, E_2 \rangle$ such that $s_1\ B\ s_2$.

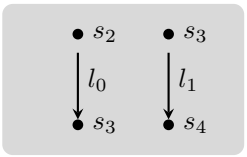
Proposition 7. *B is a bisimulation between $\langle S_1, E_1 \rangle$ and $\langle S_2, E_2 \rangle$ if and only if B is a bisimulation between the L-labelled execution coalgebras $\langle S_1, \mathsf{fun}\, E_1 \rangle$ and $\langle S_2, \mathsf{fun}\, E_2 \rangle$.*

4.3 Abrahamson Systems and Coalgebras

Informally, we can explain bisimilarity of states of labelled execution systems in the same way as we did in the case of labelled transition systems. Only now, paths are not implicitly inferred from a transition relation, but explicitly stipulated as part of the system structure. And this can have some peculiar side effects.

For example, consider two $\{l_0, l_1\}$-labelled execution systems, whose executions are as depicted in the following left and right frames respectively:





s_0 and s_2 are not bisimilar, simply because there is an execution starting from s_3, and no execution starting from s_1. But why should we care if there is? The only execution starting from s_2 has only one step, and is in perfect agreement with the only execution starting from s_0. So, intuitively, there is no difference in branching potential between the two states. We must therefore conclude that bisimilarity is, in this case, inconsistent with our informal sense of equivalence of branching structure.

A plausible remedy for this would, informally, be the following: for any path beginning at a given state, discount any branch off that path that is not a suffix of another path beginning at that same state. And indeed, this would work for this particular case. But there are more problems.

Consider two $\{l_0, l_1, l_2\}$-labelled execution systems, whose executions are those depicted in the following left and right frames respectively:

s_0 and s_5 are bisimilar, but intuitively, there is difference in branching potential between the two states: the two executions starting from s_0 diverge right away at s_0, whereas those starting from s_5 diverge after the first step at s_6. Of course, the explanation here is that there is no execution starting from s_6, and so, conceptually, the choice between the diverging steps is already made at s_5. But then, what is the point of having the two executions share the state s_6?

By now, the reader should begin to suspect what the source of our problems is: what we have called "state" in our systems does not behave as such. In a type of system supposed to serve as a modelling device for processes of some kind, it is essential that "the future behavior depends only upon the current state, and not upon how that state was reached". And this is not always the case here. What we need to do is constrain the structure of our systems so that it is.

Of course, this idea is not new. The quote above is from [22, p. 176], where Lamport required that the set of paths in a path structure be *suffix closed*, in the sense that for any path in the set, any suffix of that path is again a path in the set. It was later observed in [13] that this is not enough: one must also require that the set of paths be *fusion closed*, in the sense that for any prefix of a path in the set, and any suffix of another path in the set, if the former ends at the state at which the latter begins, then their fusion at that state is again a path in the set (see [35]). And apparently, it was Abrahamson, in [1], that first considered path structures that satisfied both requirements (see [9]).

We now adapt these requirements to our own setting. For generality, we work on the coalgebra side of the theory.

We say that $\langle C, \varepsilon \rangle$ *is Abrahamson* if and only if the following are true:

(i) for every c, l, c', and e', if $c \rhd_\varepsilon \langle\langle l, c' \rangle\rangle \cdot e'$, then $c' \rhd_\varepsilon e'$;

(ii) for every c, l, c', e'_1, and e'_2, if $c \rhd_\varepsilon \langle\langle l, c' \rangle\rangle \cdot e'_1$ and $c' \rhd_\varepsilon e'_2$, then $c \rhd_\varepsilon \langle\langle l, c' \rangle\rangle \cdot e'_2$.

Here, (i) corresponds to suffix closure, and (ii), assuming (i), to fusion closure.

We consider the class of all Abrahamson systems, defined in a similar manner, to be the largest class of "well behaved" labelled execution systems that could be useful as semantic models of behaviour. It is then quite pleasing that the full subcategory of L-**LEC** comprised of all Abrahamson L-labelled execution coalgebras is a $(\mathsf{Pow} \circ \mathsf{Seq} \circ (L \times \mathsf{Id}))$-covariety, what implies the existence of a final coalgebra among them (see [23]).

4.4 Underlying Labelled Transition Systems and Coalgebras

In an Abrahamson system, there is a clear notion of a "possible next step" relation, which induces the construction of an associated, or better, underlying labelled transition system. From a mathematical standpoint, this construction makes sense for a non-Abrahamson system as well, and is most conveniently carried out on the coalgebra side of the theory.

Assume a class C.

We write $\eta(C)$ for a class function from $\mathsf{Pow}\,\mathsf{Seq}(L \times C)$ to $\mathsf{Pow}(L \times C)$ such that for every $S \in \mathsf{Pow}\,\mathsf{Seq}(L \times C)$,

$$\eta(C)(S) = \{\mathsf{head}\, s \mid s \in S \text{ and } s \neq \langle\ \rangle\}.$$

Our choice of notation here is not arbitrary. We think of η as an operator that assigns to every class C a class function from its image under $\mathsf{Pow} \circ \mathsf{Seq} \circ (L \times \mathsf{Id})$ to its image under $\mathsf{Pow} \circ (L \times \mathsf{Id})$. And what is interesting about this operator is that for every class function $f : C_1 \to C_2$,

$$\eta(C_2) \circ \mathsf{Pow}\,\mathsf{Seq}(L \times f) = \mathsf{Pow}(L \times f) \circ \eta(C_1).$$

Thus, η is a natural transformation from $\mathsf{Pow} \circ \mathsf{Seq} \circ (L \times \mathsf{Id})$ to $\mathsf{Pow} \circ (L \times \mathsf{Id})$.

The reason why it is of interest to us here that η is a natural transformation is a theorem by Rutten, according to which, every natural transformation ν from an endofunctor F_1 on **Class** to an endofunctor F_2 on **Class** induces a functor from F_1-**Coalg** to F_2-**Coalg** that assigns to every F_1-coalgebra $\langle C, \gamma \rangle$ the F_2-coalgebra $\langle C, \eta(C) \circ \gamma \rangle$, and to every homomorphism h from an F_1-coalgebra $\langle C_1, \gamma_1 \rangle$ to an F_1-coalgebra $\langle C_2, \gamma_2 \rangle$ that same class function h, which is now a homomorphism from the F_2-coalgebra $\langle C_1, \eta(C_1) \circ \gamma_1 \rangle$ to the F_2-coalgebra

$\langle C_2, \eta(C_2) \circ \gamma_2 \rangle$ (see [36, thm. 15.1]). In a word, the induced functor preserves homomorphisms, and as a consequence, bisimulations too (see [36, lem. 5.3] and [15, thm. 5.11]).

In our case, the functor induced by η is a forgetful functor, which, informally, keeps only the first step, if any, from any execution starting from any state, and discards the rest.

Proposition 8. *If h is a homomorphism from $\langle C_1, \varepsilon_1 \rangle$ to $\langle C_2, \varepsilon_2 \rangle$, then h is a homomorphism from the L-labelled transition coalgebra $\langle C_1, \eta(C_1) \circ \varepsilon_1 \rangle$ to the L-labelled transition coalgebra $\langle C_2, \eta(C_2) \circ \varepsilon_2 \rangle$.*

Proposition 9. *If B is a bisimulation between $\langle C_1, \varepsilon_1 \rangle$ and $\langle C_2, \varepsilon_2 \rangle$, then B is a bisimulation between the L-labelled transition coalgebras $\langle C_1, \eta(C_1) \circ \varepsilon_1 \rangle$ and $\langle C_2, \eta(C_2) \circ \varepsilon_2 \rangle$.*

Of course, we can translate all this back to the system side of the theory.

Assume a binary relation $E : S \leftrightarrow \mathscr{S}(L \times S)$.

We write $\mathsf{trans}\, E$ for a binary relation between S and $L \times S$ such that for any $s \in S$ and any $\langle l, s' \rangle \in L \times S$,

$$s \; (\mathsf{trans}\, E) \; \langle l, s' \rangle \iff \text{there is } e \text{ such that } s \; E \; e,\; e \neq \langle\ \rangle, \text{ and } \mathsf{head}\, e = \langle l, s' \rangle.$$

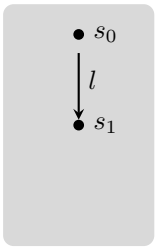
Proposition 10. $\mathsf{trans}\, E = \mathsf{rel}(\eta(S) \circ \mathsf{fun}\, E)$.

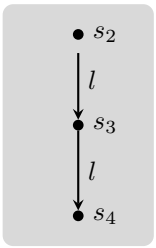
Proposition 11. *If B is a bisimulation between $\langle S_1, E_1 \rangle$ and $\langle S_2, E_2 \rangle$, then B is a bisimulation between the L-labelled transition systems $\langle S_1, \mathsf{trans}\, E_1 \rangle$ and $\langle S_2, \mathsf{trans}\, E_2 \rangle$.*

4.5 Generable Systems and Coalgebras

The converse of Proposition 11 is of course false. But it is instructive to see exactly where it fails. We go over it through a series of simple examples.

First, suppose that $\langle S_1, E_1 \rangle$ and $\langle S_2, E_2 \rangle$ are two $\{l\}$-labelled execution systems, whose single executions are as depicted in the following left and right frames respectively:

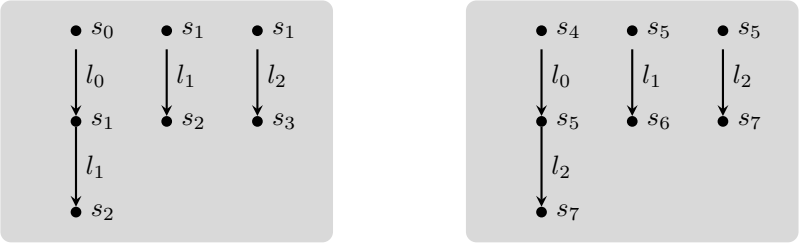




Then s_0 and s_2 are bisimilar among the two $\{l\}$-labelled transition systems $\langle S_1, \mathsf{trans}\, E_1\rangle$ and $\langle S_2, \mathsf{trans}\, E_2\rangle$, but not among $\langle S_1, E_1\rangle$ and $\langle S_2, E_2\rangle$.

The problem is easy to spot here. The two systems have one execution each. But whereas the execution of the first system has only one step, the execution of the second has two. And that second step is dropped during the labelled transition system construction.

Now suppose that $\langle S_1, E_1\rangle$ and $\langle S_2, E_2\rangle$ are two $\{l_0, l_1, l_2\}$-labelled execution systems, whose executions are as depicted in the following left and right frames respectively:



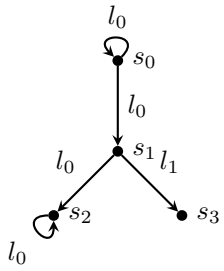
Then s_0 and s_4 are bisimilar among the $\{l_0, l_1, l_2\}$-labelled transition systems $\langle S_1, \mathsf{trans}\, E_1\rangle$ and $\langle S_2, \mathsf{trans}\, E_2\rangle$, but not among $\langle S_1, E_1\rangle$ and $\langle S_2, E_2\rangle$.

Here the problem is of a different nature. Every step of every execution is accounted for in the underlying labelled transition systems. However, the two longer executions, starting from s_0 and s_4 respectively, disagree on their second step, and that disagreement is masked by the agreement of executions starting from s_1 and s_5 respectively.

These two examples were specially chosen to target the two defining clauses of the Abrahamson property. Specifically, and informally, the systems in the first example are not suffix closed, thus violating clause (i) of the property, whereas those in the second are not fusion closed, thus violating clause (ii). Overall, none of them is Abrahamson. And since our construction was based on the idea of a "possible next step" relation, which, in the case of a non-Abrahamson system, is a conceptually ambiguous notion, it is no surprise that non-bisimilar states turn bisimilar in the constructed systems.

With Abrahamson systems, things get much more interesting.

Consider then the $\{l_0, l_1\}$-labelled transition system, with $l_0 \neq l_1$, portrayed in the following diagram:



The first $\{l_0, l_1\}$-labelled execution system that we wish to consider here is the unique Abrahamson system whose executions starting from s_0 correspond to all maximal paths in this diagram. The second is the one whose executions are all the executions of the first, except the single infinite execution stuttering around s_0. And because of this exception, s_0 is not bisimilar with itself among the two systems.

This beautiful example is from [3]. Here, it is perhaps convenient to think of the two systems as modelling the behaviour of two distinct processes, both initialized at s_0. The first process will either loop around s_0 forever, or iterate through it for a finite, indeterminate number of times before progressing to s_1. From there on, a single indeterminate choice will decide its fate. The second process, on the other hand, is not allowed to loop around s_0 forever. It must eventually advance to s_1, from where on it behaves just like the first one. What sets the behaviour of the two processes apart is, of course, the infinite stuttering around s_0, permitted for the first process, but not the second. However, this is something that cannot be determined by the sequences of actions that the two processes perform in the course of their executions, for the trace of that infinite stuttering is matched by that of every infinite execution that eventually loops around s_2. And yet the two processes ought to be distinguished. For during that infinite stuttering, the first process may always choose to branch off to a state from which it can perform l_1, whereas, in every execution having that trace, the second must eventually reach a state from which it cannot ever do so.

With respect to the failure of the converse of Proposition 11, the problem here is the existence of an infinite path in the diagram that does not correspond to any execution of a system, but whose every finite prefix is a prefix of another path that does.

This too is something that has already come up in the investigation of path structures in temporal logic. In [13], Emerson called a set of paths *limit closed* provided that for every infinite, strictly increasing chain of finite prefixes of paths in the set, the limit of that chain, in the standard topology of sequences, is again a path in the set. This property was apparently also first considered in [1]. But it was Emerson in [13] who proved the independence of all three closure properties, and the equivalence of their conjunction to the existence of a transition relation generating the given set of paths. Apart from the absence of labels, which has no bearing in this particular discussion, Emerson's setup was different in that paths

were always infinite. But this too is of no importance in our examples, which, in light of Emerson's result, appear to implicate violation of limit closure in the failure of the underlying labelled transition system to subsume all the branching information relevant to a given Abrahamson system.

Our next example is perhaps the most curious one.

Consider the simple $\{l\}$-labelled transition system portrayed in the following diagram:

There are exactly three Abrahamson $\{l\}$-labelled execution systems that one can lay over this labelled transition system. The first is the one whose only execution corresponds to the only infinite path in the diagram. The second is the one whose executions correspond to all finite paths in the diagram. And of course, the third is the one whose executions are all executions of the first and second system. But s is not bisimilar with itself among any two of the three.

Informally, the second system is not limit closed, and this is one part of the problem. But the first and third are, and so there must be something more going on here. The answer is in the difference between Emerson's setup and ours mentioned earlier. Here, executions are not always infinite. In a system that is, informally, suffix closed, if there is a finite execution, then there is an empty execution. And an empty execution creates a type of branching that is impossible to mimic in a labelled transition system.

In an Abrahamson system that is used to model the behaviour of a process, an empty execution can be used to model termination. But if there is another, non-empty execution starting from the same state, then termination becomes a branching choice, one that does not show up in the "possible next step" relation of the system. This feature of *indeterminate termination*, as we might call it, can seem a little odd at first, but is really a highly versatile mechanism, particularly useful in modelling idling in absence of input stimuli.

Finally, consider the labelled transition system of the following, trivial diagram:

There are exactly two labelled execution systems that one can lay over this labelled transition system: one that has one execution, the empty execution, and one that has no execution. And of course, s is not bisimilar with itself among the two.

This degenerate case deserves little comment. We only remark that in a suffix closed system, if a state has no execution starting from it, then it has no execution going through it.

At this point, we have found five possible causes of failure for the converse of Proposition 11. We have chosen our examples carefully, to examine each of the five separately and independently from one another. And we have observed how each of the first three connects to violation of one of the three closure properties that have been shown to collectively characterize sets of infinite paths generable by a transition relation. But finite paths add another dimension to the problem, rendering Emerson's characterization result obsolete. What we will show next is that impossibility of indeterminate termination, along with a non-triviality condition guarding against the occurrence of an isolated state, can be added to the conditions of suffix, fusion, and limit closure, to produce a complete characterization of system generability, insensitive to the length of the executions.

First, we need to make the notion of generability precise. For generality, we transfer ourselves again to the coalgebra side of the theory.

Assume a class function $\tau : C \to \mathsf{Pow}(L \times C)$, $c \in C$, and $e \in \mathsf{Seq}(L \times C)$.

We say that e *is a* τ*-orbit of* c if and only if the following are true:

(i) one of the following is true:

(1) $\tau(c) = \emptyset$ and $e = \langle\ \rangle$;

(2) there is l, c', and e' such that $\langle l, c' \rangle \in \tau(c)$ and $e = \langle \langle l, c' \rangle \rangle \cdot e'$;

(ii) for every $n \in \omega$, if $\mathsf{tail}^n\, e \neq \langle\ \rangle$, then one of the following is true:

(1) there is l and c' such that $\tau(c') = \emptyset$ and $\mathsf{tail}^n\, e = \langle \langle l, c' \rangle \rangle$;

(2) there is l, c', l', c'', and e'' such that $\langle l', c'' \rangle \in \tau(c')$ and $\mathsf{tail}^n\, e = \langle \langle l, c' \rangle \rangle \cdot \langle \langle l', c'' \rangle \rangle \cdot e''$.

We write $\mathsf{gen}\,\tau$ for a class function from C to $\mathsf{Pow}\,\mathsf{Seq}(L \times C)$ such that for any $c \in C$,

$$(\mathsf{gen}\,\tau)(c) = \{e \mid e \in \mathsf{Seq}(L \times C) \text{ and } e \text{ is a } \tau\text{-orbit of } c\}.$$

Assume a class function $\varepsilon : C \to \mathsf{Pow}\,\mathsf{Seq}(L \times C)$.

We say that τ *generates* ε if and only if $\mathsf{gen}\,\tau = \varepsilon$.

We say that ε *is generable* if and only if there is a class function from C to $\mathsf{Pow}(L \times C)$ that generates ε.

We say that $\langle C, \varepsilon \rangle$ *is generable* if and only if ε is generable.

Assume class functions $\tau_1, \tau_2 : C \to \mathsf{Pow}(L \times C)$.

Proposition 12. *If* $\tau_1 \neq \tau_2$*, then* $\mathsf{gen}\,\tau_1 \neq \mathsf{gen}\,\tau_2$.

Proposition 13. *The following are true:*

(a) $\eta(C) \circ \mathsf{gen}\,\tau = \tau$*;*

(b) if ε is generable, then $\varepsilon = \mathsf{gen}(\eta(C) \circ \varepsilon)$.

Theorem 1. *ε is generable if and only if the following are true:*

(a) for every c, l, c', and e', if $\langle\langle l, c'\rangle\rangle \cdot e' \in \varepsilon(c)$, then $e' \in \varepsilon(c')$;

(b) for every c, l, c', e'_1, and e'_2, if $\langle\langle l, c'\rangle\rangle \cdot e'_1 \in \varepsilon(c)$ and $e'_2 \in \varepsilon(c')$, then $\langle\langle l, c'\rangle\rangle \cdot e'_2 \in \varepsilon(c)$;

(c) for any $c \in C$ and every infinite sequence s, if for every $n \in \omega$, there is $e \in \varepsilon(c)$ such that for every $k < n + 1$, $\mathsf{tail}^k\, e \neq \langle\ \rangle$ and

$$\mathsf{head}\,\mathsf{tail}^k\, s = \mathsf{head}\,\mathsf{tail}^k\, e,$$

then $s \in \varepsilon(c)$;

(d) for every c and e, if $e \in \varepsilon(c)$ and $\langle\ \rangle \in \varepsilon(c)$, then $e = \langle\ \rangle$;

(e) for any $c \in C$, $\varepsilon(c) \neq \emptyset$.

Clause (a) of Theorem 1 corresponds to suffix closure, clause (b), conditioned on (a), to fusion closure, and clause (c) to limit closure. Clause (d) asserts the impossibility of indeterminate termination. Finally, clause (e) is the non-triviality condition discussed earlier, and essentially substitutes for Emerson's *left totality* condition on the generating transition relation (see [13]).

Each of these five properties has come about in connection with a different cause of failure of the converse of Proposition 11, and hence of Proposition 8 and 9. And if we have been thorough enough, we should expect that the conjunction of all five properties be sufficient a condition for eliminating that failure altogether. This turns out to be the case.

Theorem 2. *If $\langle C_1, \varepsilon_1\rangle$ and $\langle C_2, \varepsilon_2\rangle$ are generable, then h is a homomorphism from $\langle C_1, \varepsilon_1\rangle$ to $\langle C_2, \varepsilon_2\rangle$ if and only if h is a homomorphism from the L-labelled transition coalgebra $\langle C_1, \eta(C_1) \circ \varepsilon_1\rangle$ to the L-labelled transition coalgebra $\langle C_2, \eta(C_2) \circ \varepsilon_2\rangle$.*

Corollary 1. *If $\langle C_1, \varepsilon_1\rangle$ and $\langle C_2, \varepsilon_2\rangle$ are generable, then B is a bisimulation between $\langle C_1, \varepsilon_1\rangle$ and $\langle C_2, \varepsilon_2\rangle$ if and only if B is a bisimulation between the L-labelled transition coalgebra $\langle C_1, \eta(C_1) \circ \varepsilon_1\rangle$ and the L-labelled transition coalgebra $\langle C_2, \eta(C_2) \circ \varepsilon_2\rangle$.*

Once more, we can translate all this back to the system side of the theory.

Assume a binary relation $T : S \leftrightarrow L \times S$.

We write $\mathcal{E}_T(E)$ for a binary relation between S and $\mathscr{S}(L \times S)$ such that for any $s \in S$ and every $e \in \mathscr{S}(L \times S)$,

$$\begin{aligned} s\ \mathcal{E}_T(E)\ e \iff\ & \text{either there is no } \langle l, s'\rangle \\ & \text{such that } s\ T\ \langle l, s'\rangle, \text{ and } e = \langle\ \rangle, \\ & \text{or there is } \langle l, s'\rangle \\ & \text{such that } s\ T\ \langle l, s'\rangle,\ \mathsf{head}\, e = \langle l, s'\rangle, \text{ and } s'\ E\ \mathsf{tail}\, e. \end{aligned}$$

Proposition 14. $\mathcal{E}_T(E) = \mathsf{rel}\,\mathcal{G}_{\mathsf{fun}\,T}(\mathsf{fun}\,E)$.

We think of $\mathcal{E}_T$ as a function on binary relations between S and $\mathscr{P}(L \times S)$. And the reason that we are interested in this function is that it preserves the ordering of binary relations between S and $\mathscr{P}(L \times S)$ induced by the inclusion relation on their graphs: for every binary relation $E_1, E_2 : S \leftrightarrow \mathscr{P}(L \times S)$, if

$$\mathsf{graph}\,E_1 \subseteq \mathsf{graph}\,E_2,$$

then

$$\mathsf{graph}\,\mathcal{E}_T(E_1) \subseteq \mathsf{graph}\,\mathcal{E}_T(E_2).$$

This ordering is of course a complete lattice, and hence, by Tarski's Lattice-theoretical Fixpoint Theorem, so is the set of all fixed points of $\mathcal{E}_T$.

We write $\mathsf{exec}\,T$ for the greatest fixed point of $\mathcal{E}_T$.

We say that T *generates* E if and only if $\mathsf{exec}\,T = E$. We say that E is generable if and only if there is a binary relation between S and $L \times S$ that generates E. We say that $\langle S, E\rangle$ *is generable* if and only if E is generable.

Proposition 15. *The following are true:*

(a) $\mathsf{trans}\,\mathsf{exec}\,T = T$*;*

(b) if E *is generable, then* $\mathsf{exec}\,\mathsf{trans}\,E = E$*.*

Proposition 16. $\langle S, E\rangle$ *is generable if and only if the* L*-labelled execution coalgebra* $\langle S, \mathsf{fun}\,E\rangle$ *is generable.*

Theorem 3. *If* $\langle S_1, E_1\rangle$ *and* $\langle S_2, E_2\rangle$ *are generable, then* B *is a bisimulation between* $\langle S_1, E_1\rangle$ *and* $\langle S_2, E_2\rangle$ *if and only if* B *is a bisimulation between the* L*-labelled transition systems* $\langle S_1, \mathsf{trans}\,E_1\rangle$ *and* $\langle S_2, \mathsf{trans}\,E_2\rangle$*.*

We would like to finish this section with a few remarks.

Proposition 15 and Theorem 3 confirm what has been implicit throughout this section: generable labelled execution systems are just another representation of labelled transition systems. This is even more evident in the coalgebra side of the theory.

We write L-$\mathbf{LEC}_{\mathrm{gen}}$ for the category whose objects are all the generable L-labelled execution coalgebras, and arrows all the homomorphisms from one generable L-labelled execution coalgebra to another.

Theorem 4. L-$\mathbf{LEC}_{\mathrm{gen}}$ *and* L-$\mathbf{LTC}$ *are isomorphic.*

Thus, for all practical purposes, generable labelled execution coalgebras are equivalent to labelled transition coalgebras.

In light of this equivalence, Theorem 1 does not just characterize generable labelled execution coalgebras. It marks the boundary between the expressive power of labelled transition coalgebras and labelled execution coalgebras. And what it implies is that there is no sense in choosing the latter over the former, unless we are willing to give up one or more of the five properties listed in the respective clauses of Theorem 1.

5 Related Work

Despite the limitations of the concept of labelled transition system hinted at in the introduction, instead of replacing transition systems with execution systems, the general trend in the process algebra community has been to augment the former with all kinds of different pieces of information that would allow one to distinguish the "admissible" sequences of transitions from the "inadmissible" ones. And more often than not, the result was a type of modelling structure that could no longer claim adherence to the observational view. The few attempts that did use executions directly, at least those that we know of (see [11], [10]), were not concerned with organizing them into structures and looking at their branching properties, and anyway, seem to have received only scant attention.

The first place where we see executions organized into structures is not process algebra, but temporal logic. These so-called "path structures" are quite popular in the beginning. We do not see a formal concept of bisimulation for them, but there is definitely interest in their branching properties. The notions of suffix, fusion, and limit closure are all defined in connection with path structures. Eventually, they give way to Kripke structures, inherited from modal logic, and claimed to provide "a setting more appropriate to concurrency" (see [14, p. 152]). They do not, we think. But despite the voiced concerns for a separation between implementation and correctness issues in reasoning about concurrent programs (e.g., see [9]), transitions remain in the lead role.

In [18], Hennessy and Stirling introduce what appears to be the first type of labelled execution system in the literature. They call systems of that type *general transition systems*, and in their definition, demand not only suffix and fusion closure, but prefix closure as well, with the justification that it "also appears to be natural" (see [18, p. 27]). They also define a concept of *extended bisimulation* for such systems, which is basically the same as our formally derived concept of bisimulation between labelled executions systems (see Definition 6). The focus in [18] is in logic, and specifically, in a generalization of Hennessy-Milner Logic (see [17]) to general transition systems. But what is surprising is that no attempt is later made to apply the ideas of general transition systems and extended bisimulations to the semantics of processes.

More than ten years later, these ideas pop up in a "very rough and incomplete draft" of Aczel (see [3]), who is aware of Hennessy's work in [16], a precursor of [18], but apparently, unaware of the work in [18] (see footnote in [3, p. 3]). Aczel's

intention is to apply the final universe approach of [2] to the semantics of Milner's SCCS with finite delay (see [26]). The proposed type of structure is a generalized type of labelled transition system, where each state is equipped with the set of all infinite sequences of transitions "admissible" from that state. An added condition of "stability" makes structures of that type ultimately equivalent to the general transition systems of [18], but only because the latter are prefix closed. Eventually, these structures are represented as coalgebras over **Class**, and [2, thm. 2.2] is used to prove the existence of a final coalgebra in the full subcategory of all such coalgebras that are "stable".

The only other place where we find these ideas applied to the semantics of processes is [19]. The starting point is again Milner's SCCS with finite delay, and the structures used are practically the same as in [3]. But the approach is purely categorical. Indeed, the main goal in [19] is showing how much can be done within category theory alone.

Comparing [18], [3], and [19] with our work here, there are two things that we think stand out and would like to mention. First, as regards the general idea underlying the concept of labelled execution system, we find that in all three of [18], [3], and [19], the notion of indeterminate termination, and its use in modelling the behaviour of reactive systems, has been completely overlooked. This is easy to put right in [18], where prefix closure is an added feature, but not so in [3] and [19], where the property is practically built into the structure of a system. And second, as regards the formalization of the idea, we believe that the present approach represents a great simplification, both conceptually and notationally, over what was done in all three of [18], [3].

It should be emphasized that the precedence of [18], [3], and [19] over our work here is not causal, only temporal. Our ideas were developed, and for the most part, worked out before any acquaintance with these studies. The above review was mainly driven by our curiosity to understand why ideas that in retrospect seem so natural have not found their way into the household of the average concurrency theorist. In the end, one can only speculate. One thing is certain though: if matters of pedagogy have played any role in this, transition systems have definitely profited from it; for people like pictures, and execution systems are impossible to draw.

6 Conclusion

As we saw in Section 4.5, in order for the use of labelled execution systems to be justified, one or more of the five properties listed in the respective clauses of Theorem 1 must be discarded. But as we saw in Section 4.3, if we want our systems to be "well behaved", we must be sure to hang on to the first two of those. Therefore, if we ignore the rather uninteresting non-triviality condition, we are left with having to give up limit closure, impossibility of indeterminate termination, or both.

In fact, giving up any of these two properties has its own merit. For example, giving up limit closure enables us to faithfully model the finite delay property, so intrinsically bound to the notion of asynchronous parallelism. Possibility of indeterminate termination, on the other hand, provides us with the means of simulating the behaviour of a capricious environment that may at any time cease to produce input stimuli.

Returning to the discussion in our introduction, it is not hard to imagine how one could use Abrahamson systems to provide a model for Milner's CCS that avoided "explicating parallelism in terms of non-determinism", and to be sure, distinguished between the two sides of (2). The critical step is of course in the treatment of parallel composition as a *fair merge* operation over the executions of the individual systems, which, however, is rather straightforward (e.g., see [30]). But if one is really serious about this endeavour, one must account for abstraction, whereby certain actions of an agent become unobservable. And this calls for a suitably weakened version of bisimilarity among labelled execution systems. An interesting question, then, is whether such a weaker version can be canonically obtained by the methods used here. This should be possible, and could, as keenly observed by some of the referees, lead through Theorem 4 to a coalgebraic characterization of weak bisimilarity among labelled transition systems as well, a goal that has heretofore remained elusive.

Acknowledgement: We would like to thank the anonymous referees for their help in improving the presentation of this work.

References

1. Karl Abrahamson. *Decidability and Expressiveness of Logics of Programs.* PhD thesis, University of Washington at Seattle, 1980.
2. Peter Aczel. Final universes of processes. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 1–28, London, UK, 1994. Springer-Verlag.
3. Peter Aczel. A semantic universe for the study of fairness. VERY ROUGH AND INCOMPLETE DRAFT, October 1996.
4. Peter Aczel and Nax Paul Mendler. A final coalgebra theorem. In *Category Theory and Computer Science*, pages 357–365, London, UK, 1989. Springer-Verlag.
5. Jiří Adámek, Stefan Milius, and Jiří Velebil. On coalgebra based on classes. *Theoretical Computer Science*, 316(1-3):3–23, 2004. Recent Developments in Domain Theory: A collection of papers in honour of Dana S. Scott.
6. Jos C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.
7. Jon Barwise and Lawrence Moss. *Vicious Circles.* Number 60 in Lecture Notes. CLSI, 1996.
8. Martin Berger. An interview with Robin Milner. http://www.informatics.sussex.ac.uk/users/mfb21/interviews/milner/, September 2003.

9. Constantin Courcoubetis, Moshe Y. Vardi, and Pierre Wolper. Reasoning about fair concurrent programs. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 283–294, New York, NY, USA, 1986. ACM.
10. Philippe Darondeau. About fair asynchrony. *Theoretical Computer Science*, 37:305–336, 1985.
11. Philippe Darondeau and Laurent Kott. On the observational semantics of fair parallelism. In Josep Diaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 147–159. Springer Berlin / Heidelberg, 1983.
12. Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
13. E. Allen Emerson. Alternative semantics for temporal logics. *Theoretical Computer Science*, 26(1-2):121–130, 1983.
14. E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, January 1986.
15. H. Peter Gumm. Elements of the general theory of coalgebras. Lecture Notes for LUATCS'99 at Rand Afrikaans University, Johannesburg, South Africa, 1999.
16. Matthew Hennessy. Axiomatising finite delay operators. *Acta Informatica*, 21(1):61–88, 1984.
17. Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, January 1985.
18. Matthew Hennessy and Colin Stirling. The power of the future perfect in program logics. *Information and Control*, 67(1-3):23–52, 1985.
19. Thomas T. Hildebrandt. A fully abstract presheaf semantics of SCCS with finite delay. *Electronic Notes in Theoretical Computer Science*, 29:102–126, 1999. CTCS '99, Conference on Category Theory and Computer Science.
20. Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.
21. Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976.
22. Leslie Lamport. "Sometime" is sometimes "not never": On the temporal logic of programs. In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '80, pages 174–185, New York, NY, USA, 1980. ACM.
23. Eleftherios Matsikoudis and Edward A. Lee. Labelled execution systems. http://chess.eecs.berkeley.edu/pubs/882.html, 2012.
24. Robin Milner. Synthesis of communicating behaviour. In *7th MFCS: Mathematical Foundations of Computer Science*, volume 64 of *Lecture Notes in Computer Science*, pages 71–83. Springer, 1978.
25. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1980.
26. Robin Milner. A finite delay operator in synchronous CCS. Technical Report CSR-116-82, University of Edinburgh, 1982.
27. Robin Milner. A calculus of communicating systems. Report ECS-LFCS-86-7, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, August 1986. (First published by Springer-Verlag as Vol.92 of Lecture Notes in Computer Science).
28. Robin Milner. *Communication and Concurrency*. Prentice Hall International Series in Computer Science. Prentice-Hall, Upper Saddle River, NJ, USA, 1989.

29. Edward F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34:129–153, 1956.
30. David Park. On the semantics of fair parallelism. In Dines Bjoorner, editor, *Abstract Software Specifications*, volume 86 of *Lecture Notes in Computer Science*, pages 504–526. Springer Berlin / Heidelberg, 1980.
31. David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer Berlin / Heidelberg, 1981.
32. Gordon D. Plotkin. A structural approach to operational semantics (Aarhus notes). Technical Report DAIMI FN–19, Computer Science Department, Aarhus University, September 1981.
33. Gordon D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:3–15, 2004.
34. Amir Pnueli. The temporal semantics of concurrent programs. In Gilles Kahn, editor, *Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin / Heidelberg, 1979.
35. Vaughan R. Pratt. Process logic: Preliminary report. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '79, pages 93–100, New York, NY, USA, 1979. ACM.
36. Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.
37. Jan J. M. M. Rutten and Daniele Turi. On the foundations of final semantics: Non-standard sets, metric spaces, partial orders. In *Proceedings of the REX Workshop on Sematics: Foundations and Applications*, pages 477–530, London, UK, 1993. Springer-Verlag.