Basic Research in Computer Science

BRICS DS-00-1 T. T. Hildebrandt: Categorical Models for Concurrency: Independence, Fairness and Dataflow

Categorical Models for Concurrency: Independence, Fairness and Dataflow

Thomas Troels Hildebrandt

BRICS Dissertation Series **DS-00-1**

ISSN 1396-7002 **February 2000**

See back inner page for a list of recent BRICS Dissertation Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `DS/00/1/`

Categorical Models for Concurrency: Independence, Fairness and Dataflow

Thomas Troels Hildebrandt

Ph.D. Dissertation

Department of Computer Science
University of Aarhus
Denmark

Categorical Models for Concurrency: Independence, Fairness and Dataflow

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfillment of the Requirements for the
Ph.D. Degree

by
Thomas Troels Hildebrandt
October 29, 1999

Abstract

This thesis is concerned with formal semantics and models for *concurrent* computational systems, that is, systems consisting of a number of parallel computing sequential systems, interacting with each other and the environment. A formal semantics gives meaning to computational systems by describing their *behaviour* in a *mathematical model*. For concurrent systems the interesting aspect of their computation is often how they interact with the environment *during* a computation and not in which state they terminate, indeed they may not be intended to terminate at all. For this reason they are often referred to as *reactive systems*, to distinguish them from traditional *calculational* systems, as e.g. a program calculating your income tax, for which the interesting behaviour is the answer it gives when (or if) it terminates, in other words the (possibly partial) function it computes between input and output. *Church's thesis* tells us that regardless of whether we choose the *lambda calculus*, *Turing machines*, or almost any modern programming language such as *C* or *Java* to describe calculational systems, we are able to describe exactly the same class of functions. However, there is no agreement on observable behaviour for concurrent reactive systems, and consequently there is no correspondent to Church's thesis. A result of this fact is that an overwhelming number of different and often competing notions of observable behaviours, primitive operations, languages and mathematical models for describing their semantics, have been proposed in the litterature on concurrency.

The work presented in this thesis contributes to a categorical approach to semantics for concurrency which have been developed through the last 15 years, aiming at a more coherent theory. The initial stage of this approach is reported on in the chapter on models for concurrency by Winskel and Nielsen in the Handbook of Logic in Computer Science, and focuses on relating the already existing models and techniques for reasoning about them. This work was followed by a uniform characterisation of behavioural equivalences from the notion of *bisimulation from open maps* proposed by Joyal, Winskel and Nielsen, which was applicable to any of the categorical models and shown to capture a large number of existing variations on bisimulation. At the same time, a class of abstract models for concurrency was proposed, the *presheaf models for concurrency*, which have been developed over the last 5 years, culminating in the recent thesis of Cattani.

This thesis treats three main topics in concurrency theory: *independence*, *fairness* and *non-deterministic dataflow*.

Our work on independence, concerned with explicitly representing that actions result from independent parts of a system, falls in two parts. The first contributes to the initial work on describing formal relationships between existing models. The second contributes to the study of concrete instances of the abstract notion of bisimulation from open maps. In more detail, the first part gives a complete formal characterisation of the relationship between two models, *transition systems with independence* and *labelled asynchronous transition systems*

respectively, which somehow escaped previous treatments. The second part studies the borderline between two well known notions of bisimulation for independence models such as 1-safe Petri nets and the two models mentioned above. The first is known as the *history-preserving bisimulation* (HPB), the second is the bisimulation obtained from the open maps approach, originally introduced under the name *hereditary history-preserving bisimulation* (HHPB) as a strengthening of HPB obtained by adding *backtracking*. Remarkably, HHPB has recently been shown to be *undecidable* for finite state systems, as opposed to HPB which is known to be decidable. We consider history-preserving bisimulation with *bounded backtracking*, and show that it gives rise to an infinite, *strict* hierarchy of *decidable* history-preserving bisimulations seperating HPB and HHPB.

The work on fairness and non-deterministic dataflow takes its starting point in the work on presheaf models for concurrency in which these two aspects have not been treated previously.

Fairness is concerned with ruling out some completed, possibly infinite, computations as *unfair*. Our approach is to give a presheaf model for the observation of *infinite* as well as finite computations. This includes a novel use of a *Grothendieck topology* to capture unique completions of infinite computations. The presheaf model is given a concrete representation as a category of *generalised synchronisation trees*, which we formally relate to the general transition systems proposed by Hennessy and Stirling for the study of fairness. In particular the bisimulation obtained from open maps is shown to coincide with their notion of *extended bisimulation*. Benefitting from the general results on presheaf models we give the first denotational semantics of Milners SCCS with finite delay that coincides with his operational semantics up to extended bisimulation.

The work on non-deterministic dataflow, i.e. systems communicating assynchronously via buffered channels, recasts dataflow in a modern categorical light using *profunctors* as a generalisation of relations. The well known causal anomalies associated with relational semantics of indeterminate dataflow are avoided, but still we preserve much of the intuitions of a relational model. The model extends the traditional presheaf models usually applied to semantics for *synchronous* communicating processes described by CCS-like calculi, and thus opens up for the possibility of combining these two paradigms. We give a concrete representation of our model as a category of (unfolded) *monotone port automata*, previously studied by Panangaden and Stark. We show that the notion of bisimulation obtained from open maps is a congruence with respect to the operations of dataflow. Finally, we use the categorical formulation of feedback using *traced monoidal categories*. A pay off is that we get a semantics of higher-order networks almost for free, using the *geometry of interaction* construction.

Acknowledgments

Thanks to

my supervisor Glynn Winskel, for his friendly attitude and inspiring approach to research,

my co-authors, in particular Vladimiro Sassone for helping me through the first difficult stage as a PhD student and Prakash Panangaden for his catching joy for life and research,

the people who work hard to make BRICS such an excellent research environment, able to host my collaborators in the first place,

the staff and students at Aarhus University for making it a place worth giving a part of your self and where everything seems possible to achieve,

fellow PhD-students and office mates during the last 4 years,

old friends who kept in contact and the new friends I made in Aarhus,

my family for their love and support, to my parents Ellen and Troels in particular for giving me such a safe life, my twin sister Rikke for being close from the very start, by brother Christian for sharing his many interests, my sister Anne and her daughter Eja for giving me a good start in Aarhus, and my brother Just for proving that impossible projects can be possible.

Finally the deepest thanks to my wife and companion in life, Sybille, who has been the one closest to me for the last 7 years, giving me so many joyful days, and when needed listening to and helping me through my worries. Her presence and love give meaning to it all.

Contents

List of Figures

Part I

Overview

Chapter 1

Introduction

The lack of a universal computational model for concurrency has led to the proposal of a large number of different and often competing mathematical models in which to describe concurrent computational systems.

This thesis contributes to a *categorical* approach to semantics for concurrency aiming at a more coherent theory for concurrency, centered around the notion of *bisimulation from open maps* and *presheaf models for concurrency*. My goal is to extend earlier work on *independence* models for concurrency, and to take the first steps towards including two central topics in concurrency, viz. *fairness* and *dataflow*, not treated before in this framework.

A result of the categorical approach will be that we can draw on the *same* general techniques to obtain and reason about models and semantics capturing *different* notions of observable behaviour and operational primitives.

Although we have only taken the first steps, we hope to convey that the categorical approach extends naturally to include both fairness and dataflow. In both cases, models suggested by more ad hoc approaches reappear in the abstract setting, only on a much stronger fundament given by the general theory developed through the last five years for presheaf models for concurrency.

1.1 Structure of the Thesis

The thesis is divided into two parts. The first part serves as an overview and the second consists of the papers I have authored (or co-authored) during my PhD studies. Chapter 1 gives a brief introduction to semantics for computational systems and concurrency in particular, covering the issues of particular relevance to the work presented in this thesis.

In Ch. 2 we give a brief survey of the categorical approach to semantics for concurrency, taking its starting point in the work of Winskel and Nielsen to which the work in this thesis contributes.

Finally, the intention of Ch. 3 is to summarise and relate the work presented in Part II, consisting of the following papers divided between the three main topics in focus of this thesis:

- **Independence**

[60] **Chapter 4: Comparing Transition Systems with Independence and Asynchronous Transition Systems.**

Co-authored with Vladimiro Sassone. The results of this paper are published in *Proceedings of 7th International Conference on Concurrency Theory*, CONCUR '96, LNCS 1119, 1996. The present version is a slightly revised version also appearing as a BRICS report, RS-96-18.

[61] **Chapter 5: Transition Systems with Independence and Multi-Arcs.**
Co-authored with Vladimiro Sassone. The results of this paper are published in *Proceedings of Partial Order Methods in Verification*, POMIV '96, Vol. 29 of Dimacs, AMS, 1996. The present version is essentially appearing as BRICS report, RS-97-10, except the introduction is shortened to avoid unnecessary repetition from Ch. 4.

[42] **Chapter 6: On Plain and Hereditary History-Preserving Bisimulation.**
Co-authored with Sibylle Fröschle. The results of this paper are published in *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, MFCS'99, LNCS. The present version is a full version available as BRICS Report, RS-99-4.

- **Fairness**

[58] **Chapter 7: A Fully Abstract Presheaf Semantics of SCCS with Finite Delay.**
The results of this paper are published in *Proceedings of the 8th conference on Category Theory and Computer Science*, CTCS'99, 1999. The present version is a full version available as BRICS Report, RS-99-28.

- **Dataflow**

[59] **Chapter 8: A Relational Model of Non-Deterministic Dataflow.**
Co-authored with Prakash Panangaden and Glynn Winskel. The results of this paper are published in *Proceedings of the 9th International Conference on Concurrency Theory*, CONCUR '98, LNCS 1466, 1998. The present version is a full version. To be submitted to a journal.

1.2 Semantics for Concurrency

A formal semantics gives meaning to computational systems by describing their *behaviour* in a *mathematical model* at an *appropriate level of abstraction*. The choice of model and level of abstraction depends among other things on the intended use of the semantics and what we wish to observe from computations.

Two classical types of semantics suitable for different uses appear in this thesis, respectively *operational* semantics and *denotational* semantics. An operational semantics always describes the behaviour of a system in a model having character of a, more or less, abstract or idealized machine. Consequently, it captures some low-level, *intensional* aspects of the behaviour which (depending on *how* abstract the machine is) can be useful in the process of implementation. It then comes along with a notion of *observable*, or *extensional* behaviour, defining when two systems looks the same to any external observer and so should be considered as *behavioural equivalent*. A minimal requirement is that two equivalent systems should not give rise to inequivalent systems if placed in the same context.

A denotational semantics describes the behaviour of a system by a *compositional* map from the syntactical descriptions of systems (i.e. the programs of the given programming language)

to a mathematical model. Two systems are taken to be equivalent if they get mapped to the same object of the model, that is, the denotational semantics is assumed to capture the extensional behaviour of systems. The word compositional refers to the fact that the behaviour of a system is described in terms of the behaviours of its subcomponents, making it possible to reason *compositionally* about systems. To support a denotational semantics often a richer mathematical structure is required of the model. This can lead to deeper understanding of the programming language at hand and strong mathematical proof techniques.

Having *both* a denotational and operational semantics for the same language, it becomes important to relate the two to each other. The denotational semantics should only equate operationally behavioural equivalent systems, which is referred to as *adequacy* of the denotational semantics. Ideally *exactly* the operationally behavioural equivalent systems are equated by the denotational semantics, in which case it is said to be *fully abstract*.

A concurrent system consists of a number of parallel computing sequential systems, interacting with each other and the environment, and possibly distributed between different locations. What is interesting to observe of their behaviour is often how they interact with the environment *during* a computation and not in which state they terminate, indeed they may not be intended to terminate at all. Systems of this kind are often referred to as *reactive systems*, a term introduced by Pnueli in [109] to distinguish them from traditional *calculational* systems for which the interesting behaviour is the answer they give when (or if) they terminate. A typical example of a concurrent reactive system is given by a system composed of a nuclear power plant running in parallel with a safety monitoring system.

For calculational systems, the observable behaviour can reasonably be taken to be the (possibly partial) function they compute between input and output and *Church's*, or *Church-Turing thesis*, formulated by Alonzo Church [30] long before the existence of computers, then tells us that all of the many different computational models proposed at that time as capturing effective computation as e.g. the *lambda-calculus*, *μ-recursive functions* and *Turing machines* are able to describe exactly the same class of functions. Today we can add almost any modern programming language, such as *C* or *Java* to the list.

However, there is no agreement on observable behaviour or operational primitives for concurrent systems, and thus no correspondent to Church's thesis. Consequently, an overwhelming number of different and often competing notions of observable behaviours, primitive operations, languages and mathematical models for describing their semantics, have been proposed in the literature on concurrency.

Below we recall Milner's *Calculus of Communicating Systems* (CCS) with its standard transition semantics and the important concept of bisimulation due to Park. They represent an important choice of operational primitives and observable behaviour for concurrency that serves as a reference throughout the thesis. We then discuss some alternative choices focusing on those related to independence, fairness and dataflow.

1.2.1 CCS and Bisimulation Equivalence

Milner's CCS is one of the pioneering approaches to semantics of concurrent reactive systems. The term *calculus* indicates that one of the main objectives in the development of CCS was to give a careful treatment of the *structural* properties of such systems, offering an *algebraic theory* for concurrency. The primary reference is the book [90] which also includes a short description of related approaches and the development leading to CCS.

Agents with finite behaviour are constructed via the basic operators of *action prefixing*,

$$\frac{}{\alpha.t \xrightarrow{\alpha}_{\mathsf{ccs}} t}, \quad \frac{t_j \xrightarrow{\alpha}_{\mathsf{ccs}} t'}{\Sigma_{i\in I} t_i \xrightarrow{\alpha}_{\mathsf{ccs}} t'} \quad (j \in I),$$

$$\frac{t_1 \xrightarrow{\alpha}_{\mathsf{ccs}} t_1'}{t_1|t_2 \xrightarrow{\alpha}_{\mathsf{ccs}} t_1'|t_2}, \quad \frac{t_2 \xrightarrow{\alpha}_{\mathsf{ccs}} t_2'}{t_1|t_2 \xrightarrow{\alpha}_{\mathsf{ccs}} t_1|t_2'}, \quad \frac{t_1 \xrightarrow{\alpha}_{\mathsf{ccs}} t_1' \quad t_2 \xrightarrow{\bar{\alpha}}_{\mathsf{ccs}} t_2'}{t_1|t_2 \xrightarrow{\tau}_{\mathsf{ccs}} t_1'|t_2'}$$

$$\frac{t \xrightarrow{\alpha}_{\mathsf{ccs}} t'}{t\backslash L \xrightarrow{\alpha}_{\mathsf{ccs}} t'\backslash L} \quad (\alpha \notin L \cup \bar{L}) \quad \frac{t[\mathsf{rec}\,\mathsf{x}.t/x] \xrightarrow{a} t'}{\mathsf{rec}\,\mathsf{x}.t \xrightarrow{a} t'}.$$

Figure 1.1: Operational semantics for CCS

non-deterministic choice, *asynchronous parallel composition*, *restriction* and *relabelling*, as described by the BNF-grammar:

$$t ::= \alpha.t \mid \Sigma_{i\in I} t_i \mid t_1|t_2 \mid t\backslash L \mid t[f],$$

where $\alpha \in Act$ is an element of a set of *actions* $Act = \mathcal{L} \uplus \{\tau\}$, where $\mathcal{L} = \mathcal{A} \uplus \bar{\mathcal{A}}$ is a set of *labels*, partioned in a set $\mathcal{A}$ of *input* actions and a corresponding set $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$ of *output* actions (or coactions), and τ is a distinguished *internal* action. The set I is a (possibly infinite) index set, $L \subseteq \mathcal{L}$ is just a subset of labels, and $f\colon \mathcal{A} \to \mathcal{A}$ is a *relabelling* function which is extended to Act by $f(\bar{a}) = \overline{f(a)}$, for $a \in \mathcal{A}$ and $f(\tau) = \tau$. As usual the summation sign is usually omitted for a unary sum, the infix summation $t_1 + t_2$ is used as short notation for the binary sum. The term $\mathbf{0}$ was introduced as a short notation for the empty sum, representing an agent with no actions.

One of the ways of introducing agents with *infinite* behaviour given in [90] is to add *process variables* and a constructor for recursion, extending the grammar by

$$t ::= \ldots \mid x \mid \mathsf{rec}\,\mathsf{x}.t,$$

where x is a process variable.

The standard operational semantics of CCS is given by a Plotkin-style *structural operational semantics* [107] as shown in Fig. 1.1, consisting of a set of syntax directed inference rules for deriving labelled transitions between agents.

Formally, a labelled transition system is a 4-tuple [1]

$$(S, s, \longrightarrow, L),$$

with S being a set of *states*, $s \in S$ a distinguished *initial state*, L a set of *labels* or *actions* and $\longrightarrow \subseteq S \times L \times S$ a *transition relation*.

For a CCS-agent t, the labelled transition system

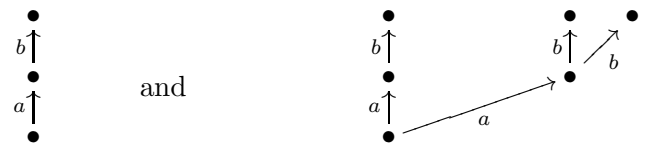
$$(S_t, t, \longrightarrow_{\mathsf{ccs}} \cap (S_t \times Act \times S_t), Act),$$

[1] Unlike in [90], we assume that transition systems have a distinguished initial state.

for $S_t = \{t' \mid t \longrightarrow_{\mathsf{CCS}}^* t'\}$ is referred to as the *derivation graph for* t. The tree obtained by ignoring repetition of states, i.e. 'unfolding' the derivation graph, is referred to as the *derivation tree for* t. In this context, labelled trees, i.e. acyclic, reachable labelled transition systems with no two transitions pointing to the same state are also referred to as *synchronisation trees* [141].

The transition semantics has a clear operational intuition: the agent $\alpha.t$ can perform an α-action and then behave like the agent t, the agent $\Sigma_{i\in I} t_i$ non-deterministically choose to behave like *one* of the agents t_i, the agent $t_1|t_2$ can either progress asynchronously by one of the agents perform an action, or perform a synchronisation action by one of the agents perform an action and the other (synchronously) the corresponding coaction. Finally, the agent $t\backslash L$ can progress as the agent t except only actions which are not in L are allowed.

However, even if one abstracts from the agents lying at the nodes, the derivation graphs or derivation trees are too detailed to be taken as observable behaviour; they do not reflect the intended *algebraic* properties of CCS. A simple example from [90] illustrating this is given by the two agents $a.b.\mathbf{0}$ and $a.b.\mathbf{0} + a.(b.\mathbf{0} + b.\mathbf{0})$ which get assigned the two, different, derivation trees



(1.1)

$a.b.\mathbf{0}$ $\qquad$ $a.b.\mathbf{0} + a.(b.\mathbf{0} + b.\mathbf{0})$

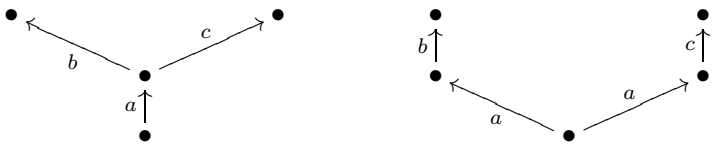
but by two applications of the intuitive equational law (which is one of the *monoid laws* [87])

$$P = P + P,$$

the second agent is easily transformed to the first:

$$\begin{aligned} a.b.\mathbf{0} + a.(b.\mathbf{0} + b.\mathbf{0}) &= a.b.\mathbf{0} + a.(b.\mathbf{0}) \\ &= a.b.\mathbf{0} + a.b.\mathbf{0} \\ &= a.b.\mathbf{0} \ . \end{aligned} \tag{1.2}$$

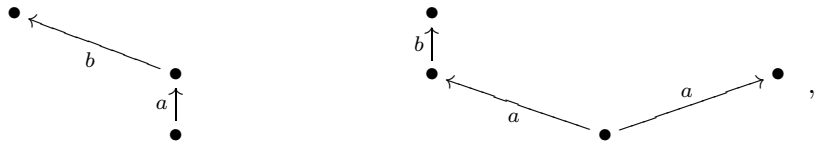
So we need a behavioural equivalence which is coarser than just isomorphism of derivation trees. A first attempt might be to regard the derivation trees as *finite automata* (taking states with no outgoing transitions as accepting states) and take language equality as behavioural equivalence, indeed the automata corresponding to the two terms above accept the same language (they have the same *linear-time* behaviour, cf. the discussion in Sec. 1.2.2 below). However the (by now standard) example from [90] given by the two agents $a.(b.\mathbf{0} + c.\mathbf{0})$ and $a.b.\mathbf{0} + a.c.\mathbf{0}$ shows that this equivalence is *too* coarse if safety properties such as deadlock should be represented correctly. The two agents get assigned the derivation trees



(1.3)

$a.(b.\mathbf{0} + c.\mathbf{0})$ $\qquad$ $a.b.\mathbf{0} + a.c.\mathbf{0}$

however by disallowing the action c (i.e. placing each agent in the context $(-)\backslash\{c\}$), we get the derivation trees



(1.4)

$a.(b.\mathbf{0} + c.\mathbf{0})\backslash\{c\}$ $\qquad$ $(a.b.\mathbf{0} + a.c.\mathbf{0})\backslash\{c\}$

which does *not* accept the same language (the second process can get stuck after having performed an a-action), i.e. language equality fails to be a *congruence* with respect to restriction.

The examples given above lead in [90] to the definition of an equivalence called *strong equivalence* which is finer than language equality but coarser than isomorphism of derivation trees. Strong equivalence is proven to be a congruence with respect to all of the operators of the calculus and allows to prove all of the algebraic laws treating the τ-action as any other action. Moreover, the meanings of (guarded) recursion are shown to be given by *unique* fixed points up to strong bisimulation in the model of synchronisation trees.

Another congruence is given which treats τ as an *unobservable* action, e.g. identifying the agents $a.\tau.\mathbf{0}$ and $a.\mathbf{0}$ and thus making more identifications than the strong congruence. This congruence is called the *observational* or *weak* congruence.

The two equivalences were noted by Park to be captured by his notion of *bisimulation* introduced (for finite automata) in [104]. Bisimulation has since become a widely used technique for defining and proving behavioural equivalences in concurrency theory. Below we recall the notion of bisimulation capturing the strong equivalence, which is referred to as *strong* bisimulation.

Definition 1.2.1 *Let $T_1 = (S_1, i_1, \longrightarrow_1, L)$ and $T_2 = (S_2, i_2, \longrightarrow_2, L)$ be labelled transition systems. A relation $\mathcal{R} \subseteq S_1 \times S_2$ is a* bisimulation *between T_1 and T_2 if*

1. *$i_1 \ \mathcal{R} \ i_2$,*
2. *if $s_1 \ \mathcal{R} \ s_2$ and $s_1 \stackrel{a}{\longrightarrow}_1 s_1'$ then there exists $s_2' \in S_2$ such that $s_2 \stackrel{a}{\longrightarrow}_2 s_2'$ and $s_1' \ \mathcal{R} \ s_2'$,*
3. *if $s_1 \ \mathcal{R} \ s_2$ and $s_2 \stackrel{a}{\longrightarrow}_2 s_2'$ then there exists $s_1' \in S_1$ such that $s_1 \stackrel{a}{\longrightarrow}_1 s_1'$ and $s_1' \ \mathcal{R} \ s_2'$.*

Two agents t and t' are then said to be *strong bisimilar* if there exists a bisimulation between their derivation graphs (or equivalently, their derivation trees). A key property of strong bisimulation is that it is *decidable* [75] for finite state systems which is useful for automatic verification.

A simple observation is that bisimulation can be defined as a relation between paths. This becomes useful when considering variations of bisimulation, as e.g. the bisimulations considered in Ch. 6.

For a transition system T, let $Seq(T)$ be the set of finite paths starting at the initial state. For $\bar{t} \in Seq(T)$, let $|\bar{t}|$ denote the *length* of the sequence and $\epsilon \in Seq(T)$ be the *empty* sequence.

Definition 1.2.2 *Let $T_1 = (S_1, i_1, \longrightarrow_1, L)$ and $T_2 = (S_2, i_2, \longrightarrow_2, L)$ be labelled transition systems. A relation $\mathcal{R} \subseteq \left\{(\bar{t}, \bar{t}') \in Seq(T_1) \times Seq(T_2) \mid |\bar{t}| = |\bar{t}'|\right\}$ is a* bisimulation *between paths of T_1 and T_2 if*

1. $\epsilon \mathcal{R} \epsilon$,

2. *if* $\bar{t}_1 \mathcal{R} \bar{t}_2$ *and* $\bar{t}_1(s_1, a, s_1') \in Seq(T_1)$ *then there exists* $\bar{t}_2(s_2, a, s_2') \in Seq(T_2)$ *such that* $\bar{t}_1(s_1, a, s_1') \mathcal{R} \bar{t}_2(s_2, a, s_2')$,

3. *if* $\bar{t}_1 \mathcal{R} \bar{t}_2$ *and* $\bar{t}_2(s_2, a, s_2') \in Seq(T_2)$ *then there exists* $\bar{t}_1(s_1, a, s_1') \in Seq(T_1)$ *such that* $\bar{t}_1(s_1, a, s_1') \mathcal{R} \bar{t}_2(s_2, a, s_2')$,

Strong bisimulation has many other useful alternative characterisations, more different to the first characterisation above. One is via *Hennessy-Milner temporal logic* [56]. Another is by *games* as in e.g. [96]. Strong bisimulation has also been shown to arise naturally in quite different settings: within *final coalgebra semantics* (e.g. [117]), and from *span of open maps* ([71]) as we will describe in Ch. 2.

1.2.2 Some Dichotomies in Concurrency Theory

As mentioned earlier, there exists many other choices of observable behaviours, primitive operations and mathematical models, than the choice represented by CCS and its standard transition semantics.

We list here a few main distinctions in concurrency, representing choices between whether or not a particular aspect of the system and its behaviour is explicitly represented in the model. We focus on the distinctions central to the work presented in this thesis. The first three are treated in [146] where more details and pointers to the literature on representative models can be found.

Behavioral versus system models. A *behavioral* model is a model that abstract from repetition of system states. *Language models* as *Hoare languages* [63] and *Mazurkiewicz trace languages* [85, 86], *synchronisation trees* [141], and *event structures* [137, 142] are all examples of behavioral models. Models that allow an interpretation of repetition of system states are referred to as *system models*. The models of *labelled transition systems* [77] and *Petri nets* [106] are prime examples of such models.

System models are most suited for giving operational semantics. The fact that a system model can offer finite descriptions of systems with infinite behaviour is exploited in various techniques for *model-checking* and automatic verification of reactive systems. For giving denotational semantics, in which infinite behaviour is often described as a limit of finite approximants, a behavioural model is usually more suitable.

Linear-time versus branching-time. A *branching-time* model is a model in which it is possible to represent *when* non-deterministic choices are resolved during a computation. Models that abstract from such information are called *linear-time* models. The language models mentioned above are typical examples of linear-time models, where the model of synchronisation trees, event structures and Petri nets are all examples of branching-time models.

In some cases, a linear-time model can be exactly what is needed. An example of this is given by Jonsson's full abstraction result [68] for non-deterministic dataflow. As we will recall in Sec. 1.2.1 below, some branching information is necessary to reflect safety conditions such as e.g. deadlock freedom in semantics of synchronously communicating systems.

Interleaving versus independence. In models like labelled transition systems or synchronisation trees, there is no way of interpreting that actions result from independent parts of the system, and thus may happen concurrently. Consequently, semantics given in these models reduce concurrency of actions to a nondeterministic choice between any of their sequential interleavings, why they are referred to as *interleaving models*. This level of abstraction is appropriate for many applications, as indicated by the wide use of CCS and related calculi equipped with transition semantics.

However, in some situations a semantics that retains information about independence may be desirable. One example of this is in connection with *action refinement* [131]. Another use of independence information is to identify *unfair* computations [102, 26]. Independence semantics have also been applied to tackle the *state space explosion problem* in automatic verification [129, 46, 48, 47], referring to the fact that there is exponentially many interleavings of a set of concurrent actions, which might not all be necessary to investigate, if the paths are known to represent concurrency.

The models of event structures, Petri nets and Mazurkiewicz trace languages mentioned above are all examples of independence models. So are the more direct extensions of transition systems as e.g. *asynchronous transition systems* ([11] and [122]) and *transition systems with independence* ([146]), which are studied in Ch. 4 and Ch. 5.

Finite versus infinite observations. In the majority of models for reactive systems, including all we have mentioned so far, infinite behaviours are completely determined by the finite behaviours. However in semantics for *concurrent* systems, it is often the case that some infinite computations are considered as inadmissible, even if all finite parts of them are admissible, referring to some kind of *fairness* condition [39]. This leads us to consider models allowing an explicit representation of infinite computations. The first approaches to semantics of fair concurrency were inspired by language theory, using ω-languages or (variations of) Büchi automata [104]. Later more direct extensions of the traditional models for finite observations were proposed, such as the model of *general transition systems* [54] which appear in Ch. 7 and the *generalised synchronisation trees* [139]. Also the semantics of fair dataflow in [68] uses sets of finite and infinite traces representing completed observations.

Global versus local interaction This distinction is essentially the one made by Abramsky in [3]. By global interaction we refer to the situation when independent agents of a concurrent system communicate via globally shared resources. A prime example are systems described in CCS-like calculi, where agents communicate via *shared names*, usually referred to as *ports*. An important point is, that when an agent performs an action, it is not a priori determined with which other agent (if any) it will interact. Local interaction refers to the situation when communication takes place through interfaces which are *local* to two fixed agents, meaning that interaction can be modelled by *composition*. Dataflow networks [74, 33] as studied in Ch. 8 are classical examples of locally interacting systems. Local interaction has been advocated lately through the *interaction semantics* programme [6, 4, 5, 3] as the right setting for studying *typed* concurrency, opening up to a *Curry-Howard paradigm* for concurrency.

Others. The list above is far from complete. Among other distinctions and aspects of behaviour are: *deterministic versus non-deterministic (or probabilistic) systems*, *asynchronous*

versus synchronous communication, *discrete versus continuous-time (or hybrid systems)*, *mobility*, ...

Variations on Bisimulation

Variations in expressiveness of the semantics, e.g. as described above, combined with a need for branching-time semantics have led to the adoption of bisimulation for a large number of variations on transition-based models. In many cases logical or game-theoretical characterisations have been adopted as well and decidability has been retained for finite state systems.

Examples of particular relevance for the work presented in this thesis are bisimulations for *independence* (or causality), *fairness* (or completed observations) and *non-deterministic dataflow* (or explicit I/O). Independence bisimulations include *pomset bisimulation*, *history-preserving bisimulation* (introduced by Rabinovich and Trakhtenbrot [115], Degano, De Nicola and Montanari [32] and Best, Devillers, Kiehn and Pomello [14] under the name of respectively *behaviour structure*, *mixed ordering* and *fully concurrent* bisimulation) and *hereditary history-preserving bisimulation* (introduced by Bednarczyk [12] and reappearing from the categorical approach described in next chapter). An interesting fact is that the hereditary history-preserving bisimulation represents one of the few bisimulations which is *undecidable* for finite state systems, as recently proved by Nielsen and Jurdzinsky [72].

Bisimulations for fairness include Parks original definition of bisimulation given for *Omega automata* [104], Milner's *fortification equivalence* [88], the *extended bisimulation* of Hennessy and Stirling [54] and a related bisimulation arising from the final co-algebra approach to semantics (Aczel [7]). As shown in Ch. 7, the extended bisimulation reappears in the categorical approach described in the next chapter.

Traditionally being described in linear-time models, there are quite few equivalences or variations of bisimulation for dataflow. Stark has given a CCS-like calculus of dataflow with a notion of *buffer bisimilarity* [127], in [120] Selinger proposes a bisimulation for transition systems with Input/Output and in Ch. 8 we give a notion of bisimulation for (unfolded) monotone port-automata.

Examples of bisimulations for models not considered in this thesis are the *timed bisimulation* in e.g. [25, 97] and the *probabilistic bisimulation* of Larsen and Skou [80].

Chapter 2

The Categorical Approach

In this chapter we give a short survey of the primary background for the thesis. This is a categorical approach to semantics for concurrency, which have been developed through the last 15 years, aiming to bring models of different expressiveness together in a more coherent theory.

The initial stage of the development is reported on in the chapter on *models for concurrency* by Winskel and Nielsen in the Handbook of Logic in Computer Science [146], and focuses on relating the already existing models and techniques for reasoning about them. In Sec. 2.1 below we recall a few results, trying to give an impression of the general idea.

A new stage began with a uniform characterisation of behavioural equivalences from the notion of *bisimulation from open maps* proposed by Joyal, Winskel and Nielsen [71], which was applicable to any of the categorical models and shown to capture a large number of existing variations on bisimulation[28, 27, 97]. In Sec 2.2 we recall the basic definitions and standard examples.

The paper [71] also suggested a class of abstract models for concurrency, the *presheaf models for concurrency*, which came with a *canonical* notion of bisimulation from open maps. The general theory of presheaf models have been developed over the last 5 years [71, 22, 143, 147, 21, 144, 19, 24, 37], culminating in the recent thesis of Cattani. We summarise some of the most important results in Sec. 2.3.

2.1 Models for Concurrency

The work reported in the chapter [146] was initiated by work of Winskel [140], followed by work of Nielsen, Rozenberg and Thiagarajan [99] and Sassone et al. [119, 118]. Focus of this work was to find formal relationships between some of the many different models existing in the literature, describing their common structure.

One of the aims declared in [146] is to set the scene for a formal classification of models for concurrency, taking as starting point some of the established distinctions of which we gave a few in the previous section and demonstrate that the categorical approach makes it possible to give a *universal* description of operators corresponding to some important constructions in process calculi. A hope was to identify the minimum categorical structure required to model such constructions.

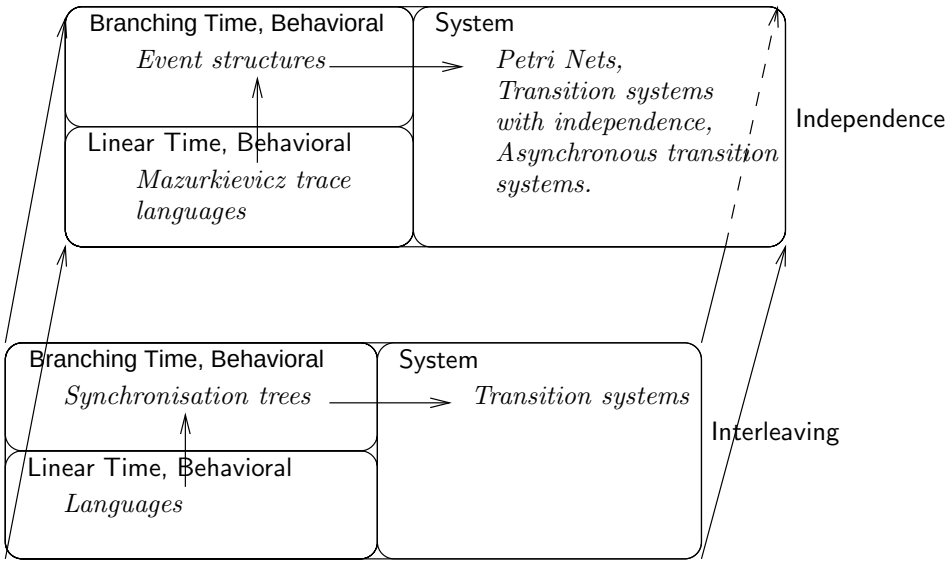


Figure 2.1: Some of the models considered in the handbook chapter and their informal classification

Being impossible to treat everything from scratch, focus was put on the three first distinctions described in Ch. 1 and a few representative models were chosen. Figure 2.1 shows some of these models and their informal classification. The arrows indicate the direction of increase in expressiveness to be expected from the discussion in Sec. 1.2.2. Another choice was to concentrate on giving a categorical description of process constructions similar to those of CCS or CSP. It was therefore natural that all the models considered were based on the central idea of having a set of atomic actions over which the behaviour of systems are defined, i.e. with which agents communicate.

The first place where category theory comes in to play is with the notion of *morphisms* between agents, [1] describing how the behaviour of one agent is related to that of another. Ensuring an associative composition of morphisms and the existence of identities makes the model into a category having agents as objects.

Remark 2.1.1 Approaches that focus on local communication as discussed in Ch. 1 naturally take processes themselves to be arrows of a category with objects representing *interface types* and composition of arrows representing communication. For fixed input and output interface, morphisms between agents typically give rise to a 2-categorical or bi-categorical structure [15, 76]. With focus on CCS-like constructions and thus *global* communication, this view was *not* taken in [146]. This is also reflected by the fact that the sets of atomic actions on which all the models treated in [146] are based were not assumed to have any structure, in particular not explicitly divided between input and output actions. The work presented Ch. 8 address how the present approach can be generalised to the 2- or bi-categorical setting.

To describe process constructions categorically, the morphisms should be able to express the relationship between an agent and its components. The approach taken to formalise the

[1] the term agent is used generally to refer to a representation of a process in a given model, e.g. a term of a calculus, a transition system etc.

structure and relationship between the different models was then to begin with the more familiar interleaving models and use this as a guide when moving to the more expressive independence models. To give a feeling of the idea we repeat here the definition of morphisms for labelled transition systems as we defined in Sec. 1.2.1.

Definition 2.1.2 *A morphism from a transition system $T = (S_T, i_T, \longrightarrow_T, L_T)$ to transition system $T' = (S_{T'}, i_{T'}, \longrightarrow_{T'}, L_{T'})$ is a pair of maps (σ, λ), where $\lambda\colon L_T \rightharpoonup L_{T'}$ is a partial map of actions and $\sigma\colon S_T \to S_{T'}$ is a map of states, such that*

- $\sigma(i_T) = i_{T'}$ *and*
- $s \xrightarrow{a}_T s'$ *implies* $\begin{cases} \sigma(s) \xrightarrow{\lambda(a)}_{T'} \sigma(s') & \text{if } \lambda(a) \text{ is defined,} \\ \sigma(s) = \sigma(s') & \text{otherwise.} \end{cases}$

We will let TS refer to the category with objects being labelled transition systems and arrows given by the definition above. There is a functor $\mathcal{L}\colon \mathsf{TS} \to \mathsf{Set}_*$, where Set_* is the category of sets and partial functions, given by

$$\begin{array}{ccc} \mathsf{TS} & & T \xrightarrow{(\sigma,\lambda)} T' \\ \downarrow\mathcal{L} & & \downarrow \\ \mathsf{Set}_* & & L_T \xrightarrow{\lambda} L_{T'} \end{array} , \tag{2.1}$$

which forms a *fibration* [13]. For a fixed set of labels, e.g. *Act*, the *fiber over Act* is the subcategory TS_{Act} induced by the transition systems with label set *Act* and (total) label preserving morphisms.

The category TS is rich in structure, allowing the interpretations of the operations: *restriction* (as (strong) cartesian liftings), *relabelling* (as (strong) cocartesian liftings), *parallel composition* (as products) and *non-deterministic choice* (as fibre coproducts). The parallel composition as interpreted by product corresponded to a "most general" parallel composition in which all conceivable synchronisations and non-synchronisations are allowed. The usual parallel operators of e.g. CCS and TCSP can be derived by such a parallel composition followed by a restriction and relabelling (reflecting a *synchronisation algebra* [138, 141]).

Being just particular labelled transition systems, the above definition of morphisms obviously restricts to labelled trees, making the model of synchronisation trees into a full subcategory of TS which we denote by ST. The inclusion $\mathsf{ST} \hookrightarrow \mathsf{TS}$ is in fact a *left adjoint* to a functor $unf\colon \mathsf{TS} \to \mathsf{ST}$ mapping a transition systems T to its *unfolding* $unf(T)$. The fact that the unfolding of a tree gives the same tree back means that the adjunction is a *coreflection*, aka the category ST is a coreflective subcategory of TS.

Similarly, a synchronisation tree S naturally projects to a language $comp(S)$ consisting of all the finite *computations* of S, i.e. the sequences of actions labelling finite paths beginning at the root of S. The model of languages can be made into a category L making this projection into a functor, which is left adjoint to a full and faithful embedding of L to ST, meaning that the adjunction is a *reflection*. The two adjunctions

$$\mathsf{L} \underset{\longrightarrow}{\overset{comp}{\underset{\perp}{\longleftarrow}}} \mathsf{ST} \underset{\longrightarrow}{\overset{unf}{\underset{\top}{\longleftarrow}}} \mathsf{TS}\,.$$

express formally how the model of languages are more abstract than the model of synchronisation trees, and how this model again is more abstract than the model of labelled transition

systems. The fact that coreflections and reflections are known to have nice preservation properties makes it possible to relate semantics given by universal constructions in different categories.

The independence models was made into categories by giving appropriate definitions of morphisms and shown to be similarly related. Moreover, the resulting categories were shown to be equally rich structure, and thus supporting the same process constructions. The step from interleaving models to independence models was also described via coreflections, e.g. the category of synchronisation trees was shown to be a coreflective subcategory of the category of labelled event structures.

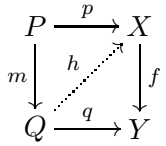
Note that for models based on a notion of events, such as asynchronous transition systems and event structures, there is a choice in whether the events are the observable actions of the system or if they express a lower level distinction and thus carry an additional labelling. This issue is also addressed in [146] and plays an important role in the work presented in Ch. 4 and Ch. 5.

2.2 Bisimulation from Open Maps

The development so far gives an account of the structural relationships between different models up to the level of (most of) the process constructions in a CCS-like language. The next stage of the development began with the paper [71] by Joyal, Nielsen and Winskel, introducing a general notion of bisimulation defined via *open maps* as studied by Joyal and Moerdijk [69]. This gives a uniform approach to bisimulation equivalences for models for concurrency, in principle applicable to any of the categorical models. In [71] it was applied to synchronisation trees and labelled event structures, in e.g. [95, 28, 100, 97] the notion of open maps is applied to a large number of other existing models.

Given a model category M (or more precisely a fiber for a fixed set of actions *Act*), the basic idea is to identify a *path category* $\mathcal{P}\colon \mathsf{P} \to \mathsf{M}$ (in [71] assumed to be a subcategory). The objects of the path category are to be thought of as the kind of behaviours that can be observed in a single observation. The arrows then express how observations relate to each other, in particular how an observation extend to a bigger observation.

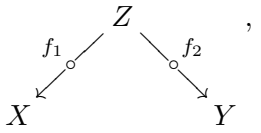
Having fixed a path category $\mathcal{P}\colon \mathsf{P} \to \mathsf{M}$, a map $f\colon X \to Y$ in M is then said to be $\mathcal{P}$*-open* (P-open if P is an inclusion, or just open if the path category is clear from the context), if it satisfies the following *path lifting* property: Whenever for two path objects P, Q of P and morphism m, p, q such that the diagram



commutes, there exists a morphism $h\colon Q \to X$ as indicated by the dotted line, making the two triangles commute. Open maps are sometimes indicated by $f\colon X \multimap Y$.

Two objects X and Y is then said to be *P-open map bisimilar* if they are related by a

span of P-open maps



assuming the model category M has pullbacks [2].

As argued in [71], it is often the case that there is a natural choice of path category, typically the subcategory of *deterministic*, i.e. conflict free processes.

For the categories $\mathsf{TS}_{\mathsf{Act}}$ and $\mathsf{ST}_{\mathsf{Act}}$ of respectively *Act*-labelled synchronisation trees and transition systems, a natural choice of observations is to take the trees consisting of just a single finite branch, i.e. the subcategory equivalent to the (partial order) category $\mathsf{Fin}_{Act} = (Act^*, \leq)$ having as objects the set of all finite sequences of actions from *Act* and arrows the usual prefix ordering. For this choice of path category, the open maps are simply the *functional bisimulations*, or *zig-zag morphisms*, and open map bisimulation indeed coincides with the classical strong bisimulation of Milner and Park.

For the category of *Act*-labelled event structures, a natural choice of observation is the subcategory of *configurations*, i.e. *conflict free*, finite event structures. This category is equivalent to a category Pom_{Act}, with objects being *pomsets*, i.e. partially ordered multi sets, over *Act*. The Pom_{Act}-open map bisimulation turns out to agree with the hereditary history-preserving bisimulation, which is the subject of study in Ch. 6.

The paper [28] shows how to capture a large number of variations on bisimulations proposed in the literature for various transition based models, e.g. Milner's weak bisimulation, Milner and Sangiorgi's barbed bisimulation and Larsen and Skou's probabilistic bisimulation as mentioned in Sec. 1.2.1. Timed bisimulation as given in e.g. [25] is characterised from open maps in [97].

2.3 Presheaf Models for Concurrency

The development so far identifies a categorical structure which is suitable to interpret the operators of a CCS-like language: Fibred Categories (over a category of (label) sets and partial functions) which have e.g. coproducts, products and pullbacks. Moreover, the open maps approach gives a uniform way of presenting bisimulation equivalence. However, there are still independent choices to be made:

- For each model the notion of simulations must be carefully defined for the resulting category to posses the right categorical structure and the presence of such needs to be proven.
- There is no a priori way of identifying the "right" notion of observation in a given model category, e.g. which are a non-trivial, adequate congruence with respect to the operations of the language.

The two extremes in choice of path category illustrates the last point above: If the model category itself is chosen as path-category the open map bisimulation will coincide with isomorphism, if the empty category is chosen as path-category *any* two objects will be open map bisimilar.

[2] If M does not have pullbacks just the existence of a sequence of spans is required, e.g. see [147].

As a possible solution to this problem, *presheaf categories* were suggested in [71] as abstract models for concurrency, known to be rich in structure and equipped with a *canonical* notion of bisimulation equivalence from open maps.

The basic idea is that the *only* choice to be made is that of a path category P of observable computations, *without* reference to any model category. We will assume that path categories always haven an *initial* object and let P^+ refer to the category obtained from P by removing the initial object(s).

Starting from a path category P, the category $\widehat{\mathsf{P}^+}$ of *presheaves* over P^+ is then interpreted as a *branching-time* model of *non-deterministic processes* with the path shapes of P.

The category $\widehat{\mathsf{P}^+}$ has as objects all functors $X\colon \mathsf{P}^{+\mathsf{op}} \to \mathsf{Set}$ (where Set is the category of all small sets and functions between them) and as arrows natural transformations between such. An important fact is that $\widehat{\mathsf{P}^+}$ is a concrete representation of the *free colimit completion* of P^+, i.e. the category obtained (up to equivalence) by freely adding all colimits to P^+. This means that any functor $F\colon \mathsf{P}^+ \to \mathsf{Q}$ for Q a cocomplete category (i.e. a category having all colimits), can be extended freely (as a left Kan extension [82]) to a (colimit preserving) functor $F_!\colon \widehat{\mathsf{P}^+} \to \mathsf{Q}$ making the diagram

$$\begin{array}{ccc} \mathsf{P}^+ & \xrightarrow{\mathcal{Y}_{\mathsf{P}^+}} & \widehat{\mathsf{P}^+} \\ & \searrow^{F} & \downarrow{\scriptstyle F_!} \\ & & \mathsf{Q} \end{array} \tag{2.2}$$

commute, where $\mathcal{Y}_{\mathsf{P}^+}\colon \mathsf{P}^+ \hookrightarrow \widehat{\mathsf{P}^+}$ is the well known *Yoneda embedding*. Writing $\mathcal{Y}^\circ_{\mathsf{P}}\colon \mathsf{P} \hookrightarrow \widehat{\mathsf{P}^+}$ for the *strict extension* of $\mathcal{Y}_{\mathsf{P}^+}$, mapping the initial object(s) of P to the empty presheaf, a map is $\mathcal{Y}^\circ_{\mathsf{P}}$-open in $\widehat{\mathsf{P}^+}$ if and only if it is surjective and $\mathcal{Y}_{\mathsf{P}^+}$-open, which is the canonical choice for open-map bisimulation in a presheaf category [71].

As shown in [71], presheaf models subsume more traditional categorical models and bisimulations.

Synchronisation trees. The category $\widehat{\mathsf{Fin}_{Act}}^+$ obtained from the path category $\mathsf{Fin}_{Act} = (Act^*, \leq)$ given in the previous section is *equivalent* to the category $\mathsf{ST}_{\mathsf{Act}}$ of Act-labelled synchronisation trees and the canonical bisimulation coincides with strong bisimulation.

Labelled event structures. The category $\widehat{\mathsf{Pom}_{Act}}^+$ obtained from the path category Pom_{Act} from the previous section has the category of Act-labelled event structures as a full subcategory (characterised abstractly in [145]), and the canonical bisimulation coincides with the *hereditary history-preserving bisimulation* for labelled event structures.

An operational reading, games and temporal logics. The construction of a synchronisation tree from a presheaf in $\widehat{\mathsf{Fin}_{Act}}^+$ can be described via the standard construction of the *category of elements* of a presheaf [83]. Since this construction apply to any presheaf category, so does the idea of representing a presheaf as a transition system as described in [147].

This does not mean that we could just as well work with transition systems. As we will see below, the rich mathematical structure of presheaf categories is indeed what makes them especially attractive. However, the operational reading can provide a helpful intuition. For

instance, it allows to give game theoretical and (tense) temporal logic characterisations of the canonical bisimulation as shown in [147].

In the work presented in Ch. 7 and Ch. 8 the operational reading was indeed a valuable key to understanding how respectively fairness and feedback could be interpreted *operationally* correct in the abstract presheaf models.

Bisimulation congruence and a domain theory for concurrency. The work on presheaf models which have followed [71], in particular the recent thesis of Cattani [24], has redeemed many of the initial promises and intuitions.

From the view of presheaf categories as a concrete representations of free colimit completions (or connected colimit completions), it follows that they can themselves can be naturally related in a category of *cocontinuous*, i.e. *colimit preserving* functors (or in the larger category of *connected colimit preserving functors*. A very important result of [24] is then that all connected colimit preserving functors between presheaf categories preserve surjective open maps with respect to the Yoneda embedding, and thus the canonical notion of bisimulation for presheaf categories. It has been shown that interpretations of the usual operators of CCS-like languages: *action-prefix*, *sum*, *restriction*, and *parallel composition* can be obtained quite elegantly from the left Kan extension as described in (2.2) above, in the case where Q is it self a presheaf category [23, 24].

Another important fact is that for any two fixed path categories P and Q, the category of colimit (or connected colimit) preserving functors between $\widehat{\mathsf{P}^+}$ and $\widehat{\mathsf{Q}^+}$ and natural transformations between them is equivalent to the presheaf category $\widehat{\mathsf{P}^{+\mathsf{op}} \times \mathsf{Q}^+}$ (or $\widehat{\mathsf{P}^{\mathsf{op}} \times \mathsf{Q}^+}$). Presheaves in $\widehat{\mathsf{P}^{+\mathsf{op}} \times \mathsf{Q}^+}$ are also referred to as *profunctors* between P^+ and Q^+, which forms the arrows of a bicategory Prof and the equivalence above gives an equivalence between Prof and the category of colimit preserving functors[15, 24]. This opens up for a *domain theory* for concurrency as explored in [22, 143, 18], replacing domains and Scott continuous functions with presheaf models and cocontinuous (or connected colimit preserving) functors.

This line of work is followed in [144], proposing a meta-language for (linear) higher-order CCS-like languages which can be given a *categorical semantics* [31] in the category of presheaf categories and connected colimit preserving functors, which seems like a natural setting in which to study *typed concurrency*.

The dataflow semantics presented in Ch. 8 is given in a subcategory of the category of connected colimit preserving functors.

Further work. Recently it has been shown in [37] how the notion of *weak bisimulation* can be described abstractly in presheaf models, exploiting the view of presheaf categories as *discrete fibrations*. In the case of synchronisation trees Milner's *observational congruence* appears naturally from the abstract definition. For the more expressive model obtained from the path category of pomsets one gets a notion of *weak hereditary history-preserving bisimulation*.

Chapter 3

Summary

The intention of this chapter is to summarize and relate the papers included in Part II. The papers all build on and contribute to the categorical approach to concurrency as summarised in the previous chapter. They are naturally divided between the three topics: Independence, fairness, and non-deterministic dataflow.

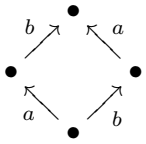
3.1 Independence

The work on independence falls in two parts. The first part consists of the papers included in Ch. 4 and Ch. 5. They contribute to the initial work on describing formal relationships between existing models, giving a complete formal characterisation of the relationship between two models, *transition systems with independence* and *labelled asynchronous transition systems* respectively, which somehow escaped previous treatments. The second part consists of the paper included in Ch. 6. It contributes to the study of concrete instances of the abstract notion of bisimulation from open maps, investigating the borderline between the *history-preserving bisimulation* and the *hereditary history-preserving bisimulation* with an emphasis on the question of decidability.

3.1.1 On the Relationship between Transition Systems with Independence and Labelled Asynchronous Transition Systems

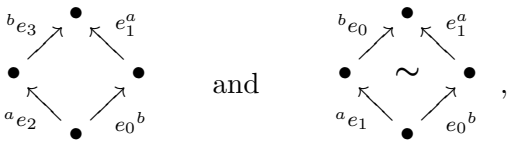
Several efforts have been devoted to the search for independence models based on the model of (labelled) transition systems, e.g., transition systems enriched with additional features that make expressing concurrency explicitly possible (cf., e.g., [111, 130, 50, 51, 133, 20]). An important motivation is to find models suitable for giving structural operational semantics to e.g. process calculi that takes *independence* information into account. The work in Ch. 4 and Ch. 5 investigate the formal relationship between two similar approaches, respectively the model of *asynchronous transition systems*, introduced independently by Bednarczyk [11] and Shields [122], and the model of *transition systems with independence*, proposed by Winskel and Nielsen [146]. Both models have been applied to give SOS style independence semantics to CCS-like calculi, respectively by Mukund and Nielsen in [93] using *labelled* asynchronous transition systems and Winskel and Nielsen in [146] using transition systems with independence. Intuitively the basic idea of these approaches is to be able to distinguish whether or

not diagrams such as



, (3.1)

represent the arbitrary interleaving of two *independent* actions, and thus the opposite facing transitions represents the 'same' action, or a choice between either an a-action that enables a b-action or vice-versa. In both models this is achieved using the idea from Mazurkiewicz trace theory in which the structure is enriching by a binary, irreflexive, symmetric *independence* relation on actions which describes their causal relationships. The model of asynchronous transition systems is obtained from the model of labelled transition systems by thinking of the labels as *events*, adding an independence relation on events and adding axioms expressing the intended meaning of events and independence: That events are deterministic, and that if two independent events can be observed in one order they can also be observed in the other (for the precise definition see e.g. [146] or Ch. 4). This approach is completely analogous to the situation in Mazurkiewicz trace languages, however it suffers in one aspect having to do with representing *branching structure* (which is not an issue in trace languages): Since events are deterministic it is not possible to represent a choice between two actions with the same label, e.g. the CCS agents $a.b.\mathbf{0} + a.c.\mathbf{0}$ and $a.(b.\mathbf{0} + c.\mathbf{0})$ from Sec. 1.2.1 cannot be distinguished. The straightforward solution to this problem, which is the one chosen in [93], is to add an *additional* labelling structure on top of the events, getting the model of *labelled asynchronous transition systems* [93, 146]. Now the two different interpretations of the transition system in (3.1) can be represented by, e.g. the two labelled asynchronous transition systems



,

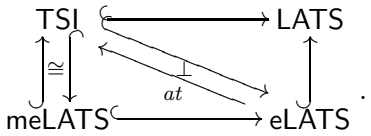
where $\sim$ indicates that the events e_0 and e_1 are independent. Although doing the job, this solution actually adds more expressiveness to the model than just what is needed to solve the above problem. For instance, it is possible to have two transitions with the same underlying event but not depending on the same transitions, e.g. we could have a diamond like the one above to the right in which the two events were *not* independent. The additional structure is not limited to these situations, e.g. the semantics given in [93] use the events to represent the low level concept of *locations* of actions, from which the independence information is derived. Consequently, if we only want to represent concurrency information, the explicit assignment of events in the model of labelled asynchronous transition systems clearly introduces some redundancies and a possible overhead in explicit identifying a notion of events. The model of transition systems with independence was proposed in [146] as a solution to this problem. Here the independence relation is imposed *directly* on the *transitions* of a 'simply-labelled' transition system and a set of axioms corresponding to those of asynchronous transition systems are then imposed, guaranteeing that a notion of events can be *derived* from the independence information, i.e. two facing edges of a diamond of independent transitions represent the same event. The semantics in [146] demonstrates how it now becomes a relatively simple task to give a SOS style independence semantics, inferring the independence information

directly from the structure of the terms without needing to introduce lower level concepts as e.g. locations. Consequently, the two different interpretations of the transition system (3.1) each has a unique representation.

Summary of our work

The derived notion of events makes the achieved expressive power of transition systems with independence comparable to that of labelled asynchronous transition systems, indeed it gives an immediate embedding of the category TSI of transition systems with independence into the category LATS of labelled asynchronous transition systems (cf. [146] or Ch. 4). The apparent differences induced on the two models by the choice of a *primitive* versus a *derived* notion of events, that seem to make them suitable for different applications, opens up the issue of investigating *formally* their analogies and differences following the line of work carried out in the first stage of the development described in the previous chapter.

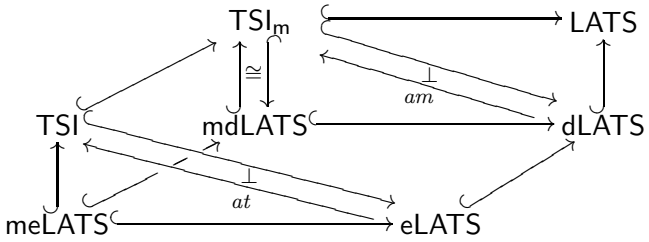
Comparing Transition Systems with Independence and Asynchronous Transition Systems. The contribution of the work presented in Ch. 4 is to give an exhaustive analysis of the relationship between the two models, which, actually, escaped the thorough analysis of models for concurrency carried out in [146, 119, 118]. We begin by looking for a functor adjoint to the obvious embedding $\mathsf{TSI} \hookrightarrow \mathsf{LATS}$ and identify the category of *extensional* labelled asynchronous transitions systems, eLATS, as the largest subcategory of LATS which admits TSI as a *coreflective* subcategory. We then prove that the category of transition systems with independence is *equivalent* to the category meLATS of so-called *event-maximal* labelled asynchronous transition systems, yielding a complete description of TSI in terms of LATS which can be useful in practise to translate back and forth between the two models when the application one has in mind requires it. The event-maximal labelled asynchronous transition systems can be seen at the same time as those transition systems that make as few identifications of transitions as possible, i.e., contain no confusion about event identities, and as those in which such identities are derivable from the independence relation, i.e., reduce the redundancy. It is worth mentioning that the converse does not hold: the asynchronous transitions systems for which the independence relation is in turn derivable from the structure of events, and therefore redundant, are slightly less general. They correspond to the transitions systems with independence for which 'independence is concurrency' considered in [119, 118]. The following commutative diagram summarize the results of the paper, making completely formal and precise the relationships between transition systems with independence and labelled asynchronous transition systems.



Transition Systems with Independence and Multi-Arcs. The starting point of work presented in Ch. 5 is the question whether or not the need to restrict to extensional asynchronous transition systems is a consequence of the intrinsic differences between the two notions of events considered. In other words, if it is necessary to assign events explicitly to be able to model situations ruled out by the extensionality constraints.

One of the situations ruled out is that of *multi-arcs*, i.e. the presence of multiple transitions with the same label between a pair of states. While being obviously redundant in the standard interleaving semantics, multi-arcs are relevant for independence semantics. For instance it allows 'quotienting' of the state-space while retaining full information about events and causality, as illustrated in Fig. 5.1.

We show that explicit events are *not* crucial to allow multiarcs. More precisely, we relax the definition of transition systems with independence by making the transitions 'first class' entities, yielding the new notion of *transition systems with independence and multi-arcs* and prove that, except for relaxing the restriction to systems withour multi-arcs, the category TSI_m of transition systems with independence and multi-arcs bears exactly the same relationships as TSI to LATS. More precisely, we prove that TSI_m is *coreflective* in the category dLATS of the *diamond-extensional* labelled asynchronous transition systems — intuitively, those transition systems that make no confusion about the identities of the events carried by transitions facing each other in independence-diamonds. Similarly to the case of TSI, dLATS is the largest subcategory of LATS for which such a result holds. Moreover, among the *diamond-extensional*, we again identify the *event-maximal* labelled asynchronous transition systems and prove that they induce the largest full subcategory of LATS, mdLATS, for which the coreflection cuts down to an *equivalence*. This yields a precise characterisation of TSI_m in terms of LATS that extends the relationships between TSI and LATS discussed above: in fact, the category of eLATS and its full subcategory meLATS are, respectively, the full subcategories of dLATS and mdLATS consisting of transition systems without multi-arcs. Summing up, the paper in Ch. 5 presents the following diagram of formal relationships between the new model of transition systems with independence and multi-arcs and labelled asynchronous transition systems.



Comments and Related Work

The analysis carried out in this paper helps in deciding when it is necessary to move to a more 'intensional' framework (a lower level of abstraction) in which further distinctions of events are introduced by assigning them explicitly. The definition of transition systems with independence and multi-arcs raises the threshold by allowing a derived notion of event also when multi-arcs are required.

As already remarked, there have been proposed several other independence models which are based on the idea of enriching transition systems or graphs. The two studied here are among the simplest, applying ideas from Mazurkievicz trace theory. The approaches in [111, 130, 50, 51, 35] use ideas from algebraic topology. Intuitively, the idea is to make the distinction between independency squares and non-independency sqaures by respresening them as respectively a *filled* square, i.e. a *surface* or an empty square, i.e. just the bounding edges. This gives a very appealing *geometric* intuition for independence and opens up for potential uses of general results from algebraic topology, cf. e.g. [35]. One limitiation of the

models we have considered, which is usually mentioned in favour for using e.g. the more geometric models, is that independence is based on a *binary* relation between transitions. This does not naturally allow us to describe the situation of having more than two actions that are pairwise independent, but cannot occur together. On a certain level of abstraction, a system of three computers trying to access two printers can be viewed as such a situation. In the geometric models, this situation is modelled very intuitively as a *hollow cube*.

The approach in [125] is to complete transition systems so they become *categories*, in which commutativity of squares can be interpreted as independency squares. These transition systems are related to the transition systems obtained from the category of elements construction for presheaves (over path-categories with independence) mentioned in Ch. 2. However, no investigation of the relationship has been carried out.

3.1.2 On Plain and Hereditary History-preserving Bisimulation

Many attempts have been made to answer the question what the appropriate generalisation of the interleaving bisimulation to independence models is. The work in Ch. 6 is a comparative study of two well known bisimulations for independence models, the *history-preserving bisimulation* (HPB) and the *hereditary history-preserving bisimulation* (HHPB). Intuitively, HPB extends the path formulation of bisimulation given in Def. 1.2.2 by requiring that any two related paths must have the "same" independence structure. As mentioned in Sec. 7.4, HPB was introduced in [115] and [32] under the name of *behaviour structure* bisimulation, and *mixed ordering* (mo) bisimulation respectively. The term *history-preserving* originates from [131], where Goltz and vanGlabbeek define the notion for event structures and prove the key property of HPB, namely that it is preserved under action refinement. This result has given history-preserving bisimulation its prominent place among independence bisimulations. In [14] the notion is introduced as fully concurrent bisimulation. There it is independently shown that HPB preserves action refinement for the more general model of Petri nets.

As recalled in the previous chapter, HHPB is the independence bisimulation obtained from the open maps approach. However, it first appeared in [12], where Bednarczyk studies several history-preserving bisimulations with a downwards closure condition. He calls sets that satisfy this condition *hereditary*. Intuitively, the downwards closure condition imposes a *backtracking* condition on the bisimulation: for any two related paths, the paths obtained by removing, i.e. backtracking, the ith transition from each path must be related too, if no later transitions in the path *causally depends* on the ith transition. The standard example showing two Petri nets which are history-preserving but not *hereditary* history-preserving bisimilar is given in Fig. 6.1.

Altogether a fair amount of work has been done already in studying both HPB and HHPB. In [132] it is suggested that HPB and HHPB may be respectively the coarsest and finest bisimulations preserving causality, branching and their interplay. However, very few attempts have been made to formally demarcate the two notions from each other. Though, as already noted in Sec. 7.4, Nielsen and Jurdzinsky have recently proven that HHPB is in fact *undecidable* for finite systems [72]. This distinguish it radically from HPB, that has been proven to be decidable for finite systems by Vogler [134]. The decidability problem for HHPB remained open for more than 5 years despite many attacks (cf. e.g. [29, 26, 57]). Indeed, one of the motivations for the work presented in Ch. 6 was to provide a stepping stone for the solution to this problem, hoping to prove *decidability*. In the light of the undecidability result, our contribution can be viewed as an investigation of the borderline between decidability and

undecidability.

Our Approach

We first characterise the difference between HPB and HHPB in *trace-theoretical* terms. In more detail, we view bisimulations between two given systems as languages over the transition set of thier synchronous product. We then show that the hereditary history-preserving bisimulations are exactly the history-preserving bisimulations which are *trace-consistent* with respect to the independence relation of the synchronous product of the two systems. This supports an intuition that HPB is (just) a bisimulation for *causality*, while HHPB is a bisimulation for *concurrency*.

We then set out to explore the frontier zone between the two notions of independence bisimulation, with an emphasis on decidability. We do this by considering hereditary history-preserving bisimulation with *bounded* backtracking. More precisely, for a fixed bound $n \in \omega$, we let (n)-HHPB refer to the hereditary history-preserving bisimulation in which backtracking is restricted to the last $n+1$ transitions. Now HHPB is the *intersection* of all of the bounded HHPBs, $(n+1)$-HHPB is obviously at least as strong as (n)-HHPB for any n, and (0)-HHPB coincides with HPB. In other words, we get an infinite hierarchy

$$\text{HHPB} \subseteq \ldots \subseteq (n+1)\text{-HHPB} \subseteq (n)\text{-HHPB} \subseteq \ldots \subseteq \text{HPB}.$$

For any fixed n, we prove that (n)-HHPB is decidable. However, we also prove that the hierarchy is *strict*. Figure 6.2 shows two concrete Petri-nets which are (n)-bounded hereditary history-preserving bisimilar but not (n+1)-bounded hereditary history-preserving bisimilar.

We end by considering subclasses of systems for which HHPB can be decided by reducing the problem to (n)-HHPB for a known n. We show that one such class is given by the systems with transitive independence relation, i.e. systems for which any two different transitions that are both independent of a third transition are independent of each other.

Comments and Related Work

There is still undiscovered territory in the zone between plain and hereditary HPB, which is studied further in work of Fröschle [40, 41]. Here interesting subclasses have been found for which the two notions can be proven to respectively coincide or not coincide.

Another interesting challenge is to find an operator which is both intuitive and useful, such as e.g. action refinement, for which HHPB but *not* HPB is a congruence.

Finally, it should be mentioned, that the trace characterisation and the decidability result of Ch. 6 is due to Sibylle Fröschle, though the idea of the decidability result was found independently and present in the work reported in [57].

3.2 Fairness

Fairness is a property of *completed* computations: a typical fairness property asserts that something (good) will *eventually* happen during the computation[1], which of course cannot be judged without observing the complete computation. As e.g. described in the book of Francez [39] there is a large number of different notions of fairness. The work presented

[1] making fairness a *liveness* property in the sense of [79]

in Ch. 7 does not in particular focus on *how* computations are deemed unfair, but follow the line of work in e.g. [54, 7] and consider models that more generally allow an explicit representation of completed computations, and thus the possibility of ruling out those which for some reason are considered inadmissible. Indeed the model of *general transition systems* proposed in [54] consists of transition systems equipped with a, suitably closed, subset of their (possibly infinite) computations, indicating which are admissible.

A benefit of the more general approach is that it allows us to study e.g. behavioural equivalences and temporal logics for fairness, independent of the choice of fairness. Of course the models should ultimately be tested against one or more fair operators to see if they allow natural interpretations. Here we focus on the extension of Milner's *Synchronous CCS* (SCCS) with a *finite delay* operator as proposed in [88]. Finite delay offers an economical way of introducing inadmissible infinite computations. Yet, it is strong enough to be able to derive an asynchronous calculus with a *fair parallel* operator guaranteeing that two processes in parallel will both eventually make progress, usually called *progress fairness*, or *concurrency fairness*.

3.2.1 Finite Delay

Syntactically the finite delay of an agent t is written ϵt. The agent ϵt can perform an unbounded number of 1-actions $\epsilon t \xrightarrow{1} \epsilon t$ (delays) but *must eventually* perform an action $\epsilon t \xrightarrow{a} t'$ if t can perform an action $t \xrightarrow{a} t'$ or *stop* if t cannot perform any actions. Defining a (possibly infinite) delay operator δ by

$$\delta t = \mathsf{rec}\,\mathsf{x}.(1 : x + t),$$

it is easy to see that ϵt has the same immediate actions as δt, in particular both agents are solutions to the equation

$$x \cong (1 : x + t). \tag{3.2}$$

This tells us that process equations will not have unique solutions as it is the case in CCS (with guarded recursion). It also suggests, as studied in [64], that one could consider a more general *fair recursion* or *finite recursion* operator $\mathsf{rec}_{\mathsf{fin}}\,\mathsf{x}.$ with the same operational semantics as the usual recursion, except that any computation with infinitely many unfoldings of the recursion is inadmissible. Then finite delay can be defined by $\epsilon t = \mathsf{rec}_{\mathsf{fin}}\,\mathsf{x}.(1 : x + t)$. This idea is consistent with the semantics given in Ch. 7.

3.2.2 Behavioural Equivalences

To deal with agents in which only some infinite computations are admissible, one must readdress the issue of how to represent the behaviour of agents. The operational semantics now in fact assigns a *general* transition system to each term, i.e. a transition system with an indication of which infinite paths are admissible. Now, since strong bisimulation will identify agents that only differ on whether some infinite computations are admissible or not (in particular ϵt is identified with δt for any term t) we need a stronger equivalence.

In [89], Milner proposes a behavioural preorder called *fortification*, which is designed such that (1) it induces an equivalence which distinguishes the two notions of delay and coincides with strong bisimulation for "standard" agents, (2) recursive processes are *least* fixed points

of the associated process equations and (3) the equivalence is a congruence with respect to all the operators of the language (assuming guarded recursion).

The two last points are as one could wish for. However although the two delays are distinguished, the fortification equivalence is too weak. As pointed out by Aczel in [7], it makes some non-desirable identifications of agents. An example of this is given by the two agents

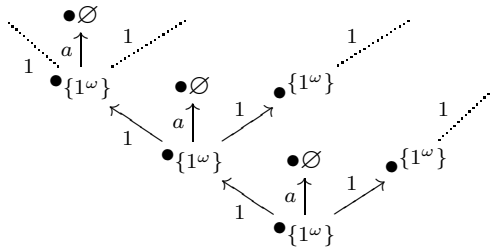
$$\delta(a{:}\mathbf{0} + \delta\mathbf{0}) \quad \text{and} \quad \epsilon(a{:}\mathbf{0} + \delta\mathbf{0}). \tag{3.3}$$

Intuitively, the two agents should not denote the same process, since the first agent can delay infinitely *remaining* able to perform an a-action, while the second agent *must* eventually reach a state in which it cannot perform an a-action. To see that these agents are identified by the fortification equivalence, it sufficient to consider an extension of the strong bisimulation which is stronger than fortification, but still identifies too much. The bisimulation we have in mind is the symmetric generalisation of fortification, extending Def. 1.2.1 by the extra conditions

1. if $s_1 \mathcal{R} s_2$ and $s_1 \xrightarrow{a_1}_1 s_{1,1} \xrightarrow{a_2}_1 s_{1,2} \ldots \xrightarrow{a_n}_1 s_{1,n} \ldots$ is an *admissible* computation path, then there exists an *admissible* computation path $s_2 \xrightarrow{a_1}_2 s_{2,1} \xrightarrow{a_2}_2 s_{2,2} \ldots \xrightarrow{a_n}_2 s_{2,n} \ldots$.

2. if $s_1 \mathcal{R} s_2$ and $s_2 \xrightarrow{a_1}_2 s_{2,1} \xrightarrow{a_2}_2 s_{2,2} \ldots \xrightarrow{a_n}_2 s_{2,n} \ldots$ is an *admissible* computation path, then there exists an *admissible* computation path $s_1 \xrightarrow{a_1}_1 s_{1,1} \xrightarrow{a_2}_1 s_{1,2} \ldots \xrightarrow{a_n}_1 s_{1,n} \ldots$.

The conditions simply say, that any two related agents must be able to produce the same set of infinite action sequences. If we label nodes of derivation trees by the sets of admissible, infinite, action sequences for the underlying agents, it is easy to see that the two agents above are identified, since they both get assigned the tree



. (3.4)

However, an intuitive strengthening will give us a bisimulation that *do* distinguish the two agents. The idea is to require, that any two corresponding states of the infinite paths in the conditions above must be in the relation too, i.e. that $s_{1,n} \mathcal{R} s_{2,n}$ for any $n \in \omega$. This is exactly the *extended bisimulation* proposed by Hennessy and Stirling in [54]. It turns out to be the canonical bisimulation arising in the presheaf model we propose in Ch. 7.

Our Approach

Our starting point is the work on presheaf models for concurrency. We have two goals: The first is to extend the categorical approach to models for infinite observations, in which to interpret fairness. The second is to give a denotational semantics for SCCS with finite delay, which agrees with the operational semantics up to a behavioural equivalence similar to the extended bisimulation given above. In the following two sections we summarise how these two goals are met.

A presheaf model for infinite observations. Recall from the previous chapter, that starting from the partial order category $\mathsf{Fin}_{Act} = (Act^*, \leq)$, the category $\widehat{\mathsf{Fin}_{Act}}^+$ is equivalent to the category ST_{Act} of Act-labelled synchronisation trees. Moreover, the typical constructions of a CCS-like language can be expressed as functors preserving the canonical equivalence.

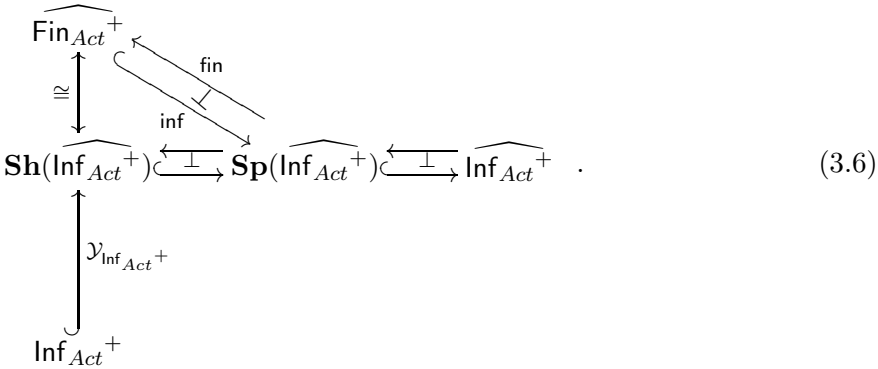
In this light, it was natural to consider the path category $\mathsf{Inf}_{Act} = (Act^* \cup Act^\omega, \leq)$ obtained simply by adding all *infinite* sequences of actions to the category Fin_{Act}. The categories $\widehat{\mathsf{Fin}_{Act}}^+$ and $\widehat{\mathsf{Inf}_{Act}}^+$ are related by two obvious functors

$$\mathsf{Fin}_{Act} \underset{\mathsf{inf}}{\overset{\mathsf{fin}}{\hookleftarrow\!\!\hookrightarrow}} \mathsf{Inf}_{Act} \quad , \tag{3.5}$$

where fin simply projects to the finite paths, and inf extends a presheaf in $\widehat{\mathsf{Fin}_{Act}}^+$ to a presheaf in $\widehat{\mathsf{Inf}_{Act}}^+$ with empty sets for any infinite path.

Assuming that X is a presheaf in $\widehat{\mathsf{Inf}_{Act}}^+$, an element $x \in X(\alpha)$ for $\alpha \in Act^\omega$ will specify a unique infinite path in the synchronisation tree corresponding to $\mathsf{fin}(\mathrm{X})$. We wish to represent that an infinite path is *admissible* by the *presence* of such a limit point, and that it is *inadmissible* by the *absence* of a limit point. With this interpretation, the model is a bit too general; it allows an infinite path to have two or even more limit points, not representing anything more than if it had only one limit point. We take the subcategory of presheaves with at most one limit point for any infinite sequence as our model, which is in fact the category $\mathbf{Sp}(\widehat{\mathsf{Inf}_{Act}}^+)$ of *separated presheaves* over $\mathsf{Inf}_{Act}{}^+$ with respect to the *canonical* Grothendieck topology for Inf_{Act}. In the standard terminology, the infinite paths and unique limit points are respectively *matching families* and *unique amalgations*.

Moreover, we can recover the category $\widehat{\mathsf{Fin}}$ within $\widehat{\mathsf{Inf}}$, as being equivalent to the category $\mathbf{Sh}(\widehat{\mathsf{Inf}})$ of *sheaves* over Inf for the same topology. In our case, a separated presheaf is a sheaf if it has *exactly* one limit point for any infinite path. Thus, a sheaf will correspond to a synchronisation tree in which *any* infinite path is admissible, i.e. a *limit closed* synchronisation tree, which is just the standard interpretation made explicit. Sheaves, separated presheaves and presheaves are known to be closely related and rich in structure [83, 148], in particular the category $\mathbf{Sh}(\widehat{\mathsf{Inf}_{Act}}^+)$ is a reflective subcategory of $\mathbf{Sp}(\widehat{\mathsf{Inf}_{Act}}^+)$, which is a reflective subcategory of $\widehat{\mathsf{Inf}_{Act}}^+$. Since the objects of $\mathsf{Inf}_{Act}{}^+$ are trivially sheaves under the Yoneda embedding we get the same, usual canonical bisimulation for all three categories. The relationships between the different categories just described are summarized in the diagram below.



(3.6)

We give a concrete representation of $\mathbf{Sp}(\widehat{\mathsf{Inf}_{Act}{}^{+}})$ as a category of *generalised synchronisation trees*, which is shown to be coreflective in a category of *generalised transition systems*, with objects being general transition systems as defined in [54] in which any finite computation is admissible. The relationship between the finitary models as described in Ch. 2 is lifted to the infinitary models as summarised in the diagram below.

$$\begin{array}{ccc} \mathbf{Sp}(\widehat{\mathsf{Inf}_{Act}{}^{+}}) & & \\ \updownarrow \cong & & \\ \mathsf{GST} & \underset{\hookrightarrow}{\overset{\leftarrow}{\top}} & \mathsf{GTS} \\ fin \dashv \updownarrow & & fin \dashv \updownarrow \\ \mathsf{ST} & \underset{\hookrightarrow}{\overset{\leftarrow}{\top}} & \mathsf{TS} \end{array} \tag{3.7}$$

Finally, we show that the extended bisimulation for generalised synchronisation trees as defined in [54] coincides via the equivalence with the canonical bisimulation obtained for $\mathbf{Sp}(\widehat{\mathsf{Inf}_{Act}{}^{+}})$.

Semantics of SCCS with finite delay. We first give an operational semantics of SCCS with finite delay in the generalised transition systems capturing exactly the definition of inadmissible computations given in terms of waiting subcomputations in [88]. We then give a denotational semantics in the presheaf model which we prove to be *equationally fully abstract* with respect to *extended* bisimulation. In this process we greatly benefit from the categorical presentation. *Unbounded non-determinism* is represented simply by (infinite) coproducts. The remaining basic operators of SCCS is interpreted as functors constructed using left Kan extensions, for which congruence properties follow almost for free. As meanings of recursion we take *final coalgebras*, corresponding to *greatest* fixed points and the finite delay operator is simply obtained as an initial algebra corresponding to a *least* fixed point of the process equation (3.2) given above. This is analogous to work on finite and infinite data-types, cf. e.g. [38].

Finally, the categorical relationships between the different models and the general theory of bisimulation from open maps reduce the problem of relating the two semantics to finding an open map within the category of generalised transition systems.

Comments and Related Work

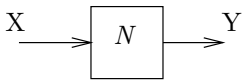
As already mentioned, our approach is closely related to the work of Hennessy and Stirling on general transition systems and extended bisimulation and the work of Aczel in []. A number of other people have proposed models for non-deterministic processes with inadmissibility of infinite computations and given denotational semantics of SCCS with finite delay and m [7, 65, 53, 54, 139]. The semantics given in [65] is also shown to be fully abstract, but with respect to the *fortification* equivalence, so it makes the non-intuitive identifications described above. Moreover, it only covers *bounded non-determinism* as obtained from terms in which only a binary sum is allowed. The semantics of [53] is also based on the fortification equivalence. Common to the models given in [65, 53, 139] is that the approximation order between agents treats infinite observations opposite than finite observations as the fortification

preorder of [88]. This means that for two related agents the largest will be the one with fewest admissible infinite computations. This ordering allows the usual interpretation of *least* fixed points as recursion, but by forcing this to be possible it seems that one introduce the counter-intuitive properties we illustrated for the fortification equivalence.

A number of questions remain to be explored. An obvious question is if one could generalise the finite delay to a *fair recursion* as in [64]. The proposed notion of open natural transformations for functors between presheaf categories should be explored further. Work is in progress on a notion of open maps between denotations of open terms stronger than the one in [22], for which open map bisimulation is a congruence with respect to recursion. The characteristic HML-like path logic [71] for extended bisimulation which can be obtained from the open maps approach should be compared to the characteristic logic given in [54]. It is worth remarking that both logics makes crucially use of tense temporal operators. Here comes the question about decidability of extended bisimulation. As remarked on in the paper by slightly altering the annotation of finite delay agents in the operational semantics an interesting subclass of agents will always be assigned *finite* (generalised) transition systems. Along a totally different line it would be interesting to explore if there is any relationship between the present approach and the more domain theoretical approaches to fairness and countable non-determinism in [108, 1]. Finally, we hope to be able to extend the presheaf model for (finitary) non-deterministic dataflow summarised in the following section to infinite observations along the lines of the present work, giving a model of dataflow in which fairness, maybe even *fair merge* [101], can be expressed.

3.3 Dataflow

Assuming some fixed set of values, which could be taken to be the set of actions Act, a dataflow network is a reactive system with a number of *input* ports and a number of *output* ports, interacting with the environment by reading and producing values on respectively its input and output ports. A network N with input ports X and output ports Y can be represented graphically as a box



,

where arrows represent a (possibly empty) collection of ports, which might be explicitly split between more arrows. The observable behaviour of a dataflow network is traditionally taken to be the input-output relation it computes between streams (possibly infinite sequences) of values from Act on respectively input and output channels. An assignment of streams to a set X of ports, i.e. a function $h\colon X \to Act^* \cup Act^\omega$, is traditionally referred to as a *history* on X and we will let H_X refer to the set of histories on X. The model of relations between such sets of histories is often referred to as the *history model*.

An essential idea of the dataflow paradigm is that a collection of networks can be combined into a single larger network, possibly connecting some of the free input and output ports as illustrated in Fig. 3.1. A connection between two ports of the same network is referred to as a *feedback* loop. Usually it is assumed that two wires cannot be connected to the same port. Note that the effect of connecting one output port to two input ports can be obtained

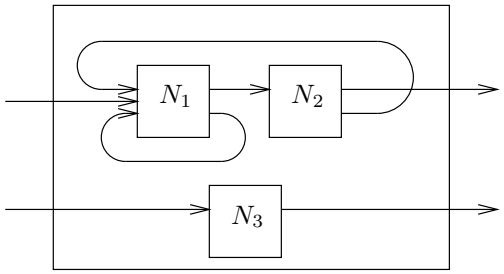
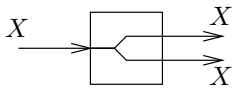


Figure 3.1: An example of a dataflow network

by assuming the existence of *fork* boxes



, (3.8)

that simply copies the values read at each input port to two output ports.

Traditionally wires are assumed to behave like unbounded FIFO-buffers and the smaller networks, sometimes referred to as *nodes*, assumed to compute asynchronously, driven by the arrival of values at their input ports. Kahn [74] observed that the behaviour of dataflow networks built from only *deterministic* nodes (and not able to test for *completion* of the input) could be captured *denotationally* in the history model, representing networks with input ports X and output ports Y as *Scott continuous* functions between *domains* of histories

$$N \colon \mathsf{H}_X \to_{cont} \mathsf{H}_Y \ , \tag{3.9}$$

ordering histories point-wise by the prefix order on streams. The key point was then, that composition could be defined as a *least fixed point* of a set of equations describing the components. This was later shown formally by several authors, e.g. Faustini [36], Lynch and Stark [81].

3.3.1 Traced Monoidal Categories and a Calculus of Boxes and Wires

Independent of *how* networks behave, we can identify a calculus of boxes and wires describing *structural* equalities (or isomorphisms) between networks, expressing that the *order* in which ports are connected does not matter. That is, two networks should behave the same if they are isomorphic as (directed) graphs with labelled nodes and ordered sets of ingoing and outgoing edges. If the basic primitives for forming networks are taken to be *sequential composition*, *parallel composition*, *crossing of wires* and *feedback* as shown in Fig. 3.2, such a calculus is naturally interpreted in a *traced symmetric monoidal category* [70]. The notion of *traced* monoidal category abstracts the notion of trace of a matrix from multi-linear algebra. However it has emerged in a variety of new contexts including the study of feedback systems [9, 3], knot theory [67] and recursion [52]. In the context of dataflow, objects of the category are interpreted as *interface* types (which might simply be a natural number representing the number of ports in the interface), arrows are interpreted as dataflow networks and sequential composition is interpreted as composition in the category. The symmetric monoidal structure is then interpreted as parallel composition and crossing of wires, and the trace is interpreted

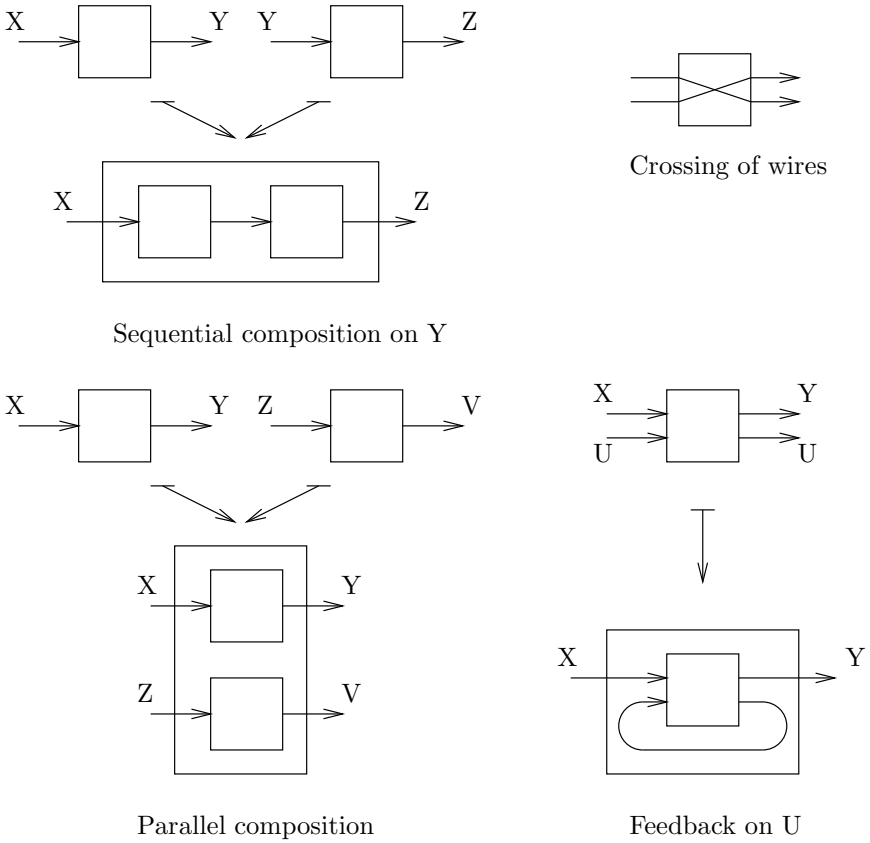


Figure 3.2: Dataflow primitives as in a Traced Symmetric Monoidal Category

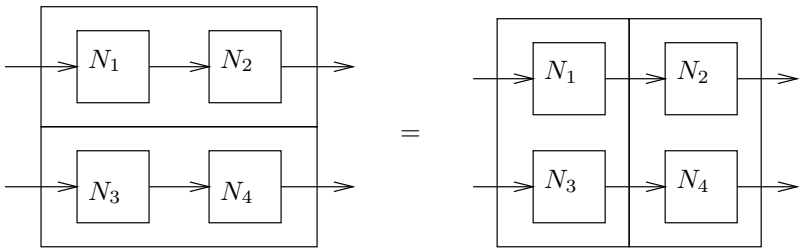


Figure 3.3: Bi-functoriality of $\otimes$ in terms of boxes and wires

as feedback loops. Recall that a (strict) monoidal category is a category C equipped with a bi-functor $\otimes\colon \mathsf{C} \times \mathsf{C} \to \mathsf{C}$ usually referred to as the *tensor product*, which is associative, and has an object I which is a left and right unit for $\otimes$ In the calculus of boxes and wires, the object I is an empty interface, i.e. an interface with no ports, and the bi-functoriality of $\otimes$ simply amounts to the intuitive equality shown in Fig. 3.3.

A monoidal category is *symmetric* if its equipped with (symmetry) isomorphisms $\sigma_{A,B}\colon A\otimes B \to A \otimes B$ natural in A and B which again are required to satisfy some commutativity conditions (which make them the right candidates for interpretations of crossings of wires). A trace in a monoidal category consists of a family of maps

$$\mathsf{Tr}^{U}_{X,Y}() : \mathsf{C}(X \otimes U, Y \otimes U) \to \mathsf{C}(X, Y),$$

required to satisfy some algebraic properties. In the calculus of boxes and wires these properties correspond to invariance under (any) rearrangement of boxes which might stretch or shorten wires but does not change or break any connections[2]. The axiomatization for traced monoidal categories as given by Joyal, Street and Verity in [70] is presented in Ch. 8, slightly simplified and specialized to strict symmetric monoidal categories as in [52].

Remark 3.3.1 The requirement of associativity of $\otimes$ and the equalities for the unit object might be relaxed to only assuming the existence of natural isomorphisms (accompanied with some commutativity conditions, cf. [82]) giving just a monoidal structure and not a strict monoidal structure. Similarly, the associativity of (sequential) composition of arrows, and the equalities of composition with identities might be relaxed (accompanied with some commutativity conditions, cf. [15]) resulting in a *bicategory* rather than a category. This is the case for one of the models for dataflow described in 8.

It is important to note that simply by being a traced (symmetric) monoidal category, a category does not necessarily allow an adequate interpretation of the behaviour of dataflow networks. An extreme example illustrating this point is provided by the (final) category with just one object and one arrow (the identity). This category is trivially a traced symmetric monoidal category, but also a quite trivial model of dataflow. A more interesting example is provided by the traced monoidal category $(\mathsf{Rel}, \times, 1, \mathsf{Tr})$ of sets and relations, with the tensor product given by the cartesian product and the trace for a relation $R \subseteq (X \times Z) \times (Y \times Z)$ given by

$$(x, z) \in \mathsf{Tr}^{Z}_{X,Y}(R) \text{ iff } \exists z \in Z.\big((x, z), (y, z)\big) \in R.$$

[2]In fact, as remarked in [121] the axioms are sound and complete for graph isomorphism for the kind of graphs mentioned in the previous footnote.

By using the equivalence $\mathsf{H}_X \times \mathsf{H}_Y \cong \mathsf{H}_{X \uplus Y}$ the traced monoidal structure of $(\mathsf{Rel}, \times, 1, \mathsf{Tr})$ restricts to the history model. However, if one considers the input/output relation $\{(h, (h_1, h_2)) \in \mathsf{H}_a \times (\mathsf{H}_b \times \mathsf{H}_c) \mid h = h_1 = h_2\}$ of the (deterministic) fork network given in (3.8) above, it is easy to see that $\mathsf{Tr}^X_{X,X}(Fork) = \mathsf{H}_b$, i.e. the relation describing the behaviour of the network that can output *any* sequence on its output port. However, for any value to be produced on the feedback wire or the output port the same value must first be read from the feedback wire. Consequently (assuming that the wire is initially empty) *no* values can be produced at the feedback wire, and so no values can be produced on the output port. In other words, the *operationally* correct trace operator should yield the *empty* relation.

Higher order structure from trace: Geometry of Interaction. The Geometry of interaction program was invented by Girard in his analysis of the fine structure of cut elimination [44, 45]. His basic insight was that higher order structure could be understood in terms of trace but this understanding was hidden in the mathematical setting - Hilbert spaces and traces of operators - that he used. In [70] Joyal, Street, and Verity and independently Abramsky [3] (see also [6]) gave the categorical expression of the idea which was that a traced monoidal category could be "completed" to yield a compact closed category. As such it gives a method for realizing higher-order constructs in terms of feedback, for more details see Sec. 8.4.2

3.3.2 Relational Semantics of Non-deterministic Dataflow

As described in the introduction, Kahn's semantics somehow marks a border between when the behaviour of dataflow systems can be described compositionally purely by its input/output relation between a domain of input values and a domain of output values and when a more detailed mathematical model is required. Brock and Ackerman [17] proved that if the non-deterministic primitive *fair-merge* was added to the calculus it was no longer possibly to give a compositional semantics in the history model. As we recall in Sec. 8.1.1 Russell [116] later showed that the history model is insufficient also if just the simpler [102] primitive *non-deterministic* choice is added to the calculus. A solution to the problem of giving a compositional semantics for non-deterministic dataflow was provided by Jonsson [68], and independently Kok [78], who gave a fully abstract denotational semantics for dataflow with fair-merge. The model of Jonsson represents the behaviour of networks by sets of (possibly infinite) sequence of input/output events, each sequence recording in which order values are read and produced on the ports during a particular, *completed* computation. In [116] Russell shows that the model of Jonsson is fully abstract even for the calculus with just non-deterministic choice and no fair primitives.

Although giving a fully abstract semantics, the solution based on language models has left the conceptually simple view of networks in Kahn's semantics in which networks are represented by functions (or relations) and feedback/network composition described by least fixed points. Moreover, where the computable behaviours in Kahn's semantics could reasonably be taken to be the (Scott) continuous functions (if one do not care about finite representations), there is no obvious way of identifying what languages in the model of Jonsson that can be realized as behaviour of (a possibly infinite) dataflow network. Consequently, it became a goal to provide a compositional semantics of non-deterministic dataflow that retains the conceptual view of processes as relations and allows network composition to be described by a kind of fixed point, generally referred to as *Kahn's principle* [81, 2].

Summary of our approach

The work on dataflow presented in Ch. 8 has two goals: To take up the challenge of giving a *relational* semantics for non-deterministic dataflow and to demonstrate how dataflow primitives, i.e. communication by composition and feedback, could be included in the framework of presheaf models for concurrency, using the notion of traced monoidal categories.

Russell's "anomaly" shows that it is already a challenge to give a relational semantics for non-deterministic networks without any kind of fair primitives, so for simplicity we consider only semantics for finite, partial computations. Consequently, in the rest of the section H_X, for a set of port names X, will refer to the partial order of functions $h\colon X \to Act^*$.

Our starting point (Sec. 8.2.2) is to give a model similar to that of Jonsson, representing networks by (prefix closed) sets of (finite) event sequences. We assume these sets are closed under permutations of independent events as in Mazurkiewicz trace languages. These processes is shown form the arrows of a traced monoidal category in which feedback is modelled *causally correct* by the trace operator. We call this category Kahn and refer to the arrows, as given by trace languages, for *Kahn-processes*.

The next step (Sec. 8.3) is to look for a traced monoidal category that is both "relational", i.e. a generalisation of the history model, and a generalisation of the category Kahn.

Having the category of (connected) colimit preserving functors mentioned in Sec. 2.3 in mind, it is natural to try replacing relations between (partially ordered sets of) histories by *profunctors*, regarding the partially ordered sets of histories as categories (Sec. 8.3). Recall that a profunctor F from H_X to H_Y, written $F\colon \mathsf{H}_X \mathrel{-\!\!\!\mapsto} \mathsf{H}_Y$, is a presheaf in $\widehat{\mathsf{H}_X^{\mathsf{op}} \times \mathsf{H}_Y}$, i.e. a bi-functor $F\colon \mathsf{H}_X \times \mathsf{H}_Y^{\mathsf{op}} \to \mathsf{Set}$.

Profunctors are a categorical generalisation of sets and relations, and as mentioned in Sec. 2.3 they form the arrows of a (bi-)category Prof, in which objects are small categories. Composition is given by a *coend* [82],

$$X;Y(\mathsf{p},\mathsf{q}) = \int^{u} X(\mathsf{p},u) \times Y(u,\mathsf{q}) \ , \tag{3.10}$$

which generalises relational composition.

We let PProf denote the subcategory induced by objects being partial order categories of histories, which we refer to as the category of *port profunctors*.

One way to illustrate how a profunctor $F\colon \mathsf{H}_X \times \mathsf{H}_Y^{\mathsf{op}} \to \mathsf{Set}$ is a categorical generalisation of a relation $F \subseteq \mathsf{H}_X \times \mathsf{H}_Y$ is by regarding the relation as a function

$$F\colon \mathsf{H}_X \to \mathcal{P}(\mathsf{H}_Y)$$

and noting that the profunctor can equivalently be regarded as a functor

$$F\colon \mathsf{H}_X \to \widehat{\mathsf{H}_Y}.$$

The "generalisation" then amounts to considering the partial orders H_X and H_Y as categories, replacing the powerset (or powerdomain) operator by the free colimit completion (cf. Ch. 2) and taking F to be a functor. It is a fact that Prof can be equipped with a tensor product and a trace, which generalises the traced monoidal structure on $(\mathsf{Rel}, \times, 1)$, and satisfy the axioms of a traced monoidal category up to isomorphism. As for Rel the traced monoidal structure restricts to PProf.

Similarly to the work on fairness, our way to investigate if PProf supports natural interpretations of dataflow was to consider the operational reading given by the category of elements construction (Sec. 8.3.2).

The encouraging result was that profunctors of the kind $F\colon \mathsf{H}_X \mathrel{-\!\!\!\mapsto} \mathsf{H}_Y{}^+$ (corresponding to connected colimit preserving functors from $\widehat{\mathsf{H}_X{}^+}$ to $\widehat{\mathsf{H}_Y{}^+}$ as mentioned in Sec. 2.3) in this way could be concretely represented as a category of (unfolded) *monotone port automata* as introduced by Stark and Panangaden [103].

The concrete representation gave an immediate functor Seq from the category of rooted port profunctors to Kahn simply mapping a rooted profunctor to the set of finite runs of its associated port automaton. This made the category PProf look promising as a "relational" model of non-deterministic dataflow.

However, the functor Seq does not preserve the trace of PProf as given by the coend. The trace suffers from being a generalisation of the trace in $(\mathsf{Rel}, \times, 1)$, indeed the (counter)example with the fork network given for Rel in the previous section carries over to PProf, showing that it is not a "causally" correct trace.

The solution was found by realising from the operational reading of profunctors as port automata that the "correct" trace appeared as a (rooted) *sub*profunctor of the trace as given by the coend in PProf, essentially obtained by removing all parts profunctor which in the associated automaton are not reachable by a *secured* computation, that is, a computation in which any token on the feedback channels appear as output before it appears as input. By restricting attention to the category $\mathsf{PProf}_\mathsf{s}$ of *stable profunctors* $\mathsf{PProf}_\mathsf{s}$, which is the category with arrows being *pullback preserving* functors $F\colon \mathsf{H}_X \to \widehat{\mathsf{H}_Y{}^+}$, the subprofunctor indeed gives a well defined trace, which is preserved by the functor $Seq\colon \mathsf{PProf}_\mathsf{s} \to \mathsf{Kahn}$. Notably, the trace can be defined, using the standard construction of the subdivision category, as the composition of two functors, the first restricting the functor to secured states, the second being the usual colimit.

As a consequence of being presheaf categories, we automatically get (Sec. 7.4) a notion of bisimulation from (surjective) open maps between stable port profunctors for a fixed choice of input and output ports. This bisimulation is proven to be a congruence with respect to parallel composition, sequential composition and feedback. In proving this we benefit from the general congruence properties mentioned in Sec. 2.3.

From the traced monoidal structure of the two categories, Kahn and $\mathsf{PProf}_\mathsf{s}$, we get models of (linear) higher-order dataflow using the Geometry of Interaction construction to construct compact closed categories $\mathcal{G}(\mathsf{Kahn})$ and $\mathcal{G}(\mathsf{PProf}_\mathsf{s})$, which we denote by HKahn and $\mathsf{HPProf}_\mathsf{s}$. However, strictly speaking the construction do not immediately apply to $\mathsf{PProf}_\mathsf{s}$. Since it is a bicategory, only satisfying the axioms of a TMC up to isomorphism. $\mathcal{G}(\mathsf{PProf}_\mathsf{s})$ will not be a category, e.g. composition is only associative up to isomorphism. One solution is consider the quotient of $\mathsf{PProf}_\mathsf{s}$ with respect to open map bisimulation, analogous to the definition of the category $\mathcal{AS}Proc$ in [4, 43]. That is, instead of $\mathsf{PProf}_\mathsf{s}$ use the category with objects being path categories as usual, but taking arrows to be equivalence classes with respect to open map bisimulation. By the congruence result, this is indeed well defined and it is easy to check that we get a traced (strict) symmetric monoidal category.

Comments on Future and Related Work

We would like to emphasise the view of the profunctor model of dataflow summarised above as an extension of the presheaf model for synchronous communication, adding asynchronous input and independence. This can be seen by first considering (stable) profunctors of the form $F\colon \mathsf{H}_\varnothing \mathrel{-\!\!\!\vdash\!\!\!\to} \mathsf{H}_1{}^+$ for 1 being the one-element set, to be interpreted as networks with *no* input and just one output port. The homcategory $\mathsf{PProf}[\mathsf{H}_\varnothing, \mathsf{H}_1]$, is trivially equivalent to the category $\widehat{\mathsf{Fin}_{Act}{}^+}$ corresponding to synchronisation trees, and the construction of port automata indeed just specialise to the one for synchronisation trees. Now we can add a set X of asynchronous input ports, considering profunctors in $\mathsf{PProf}[\mathsf{H}_X, \mathsf{H}_1]$. From such profunctors, profunctors modelling networks with more than one output port can be obtained by parallel composition.

In this sense our dataflow semantics unifies asynchrony and synchrony.

The above considerations suggest a small calculus for dataflow similar to the one in [127]. The idea is to take a pair (α, t), where t is an CCS agent with action set $\mathsf{H}_X \times Act$ and $\alpha \in \mathsf{H}_X$ is an *input buffer*, to represent agents with asynchronous input ports X and one output port. The operational semantics is then given by the rules

$$\frac{}{(\alpha, t) \overset{\mathsf{i}\,\langle x, v\rangle}{\longrightarrow} (\langle x, v\rangle\alpha, t)}, \qquad \frac{t \overset{(\alpha', b)}{\longrightarrow}_{\mathsf{ccs}} t'}{(\alpha\alpha', t) \overset{\mathsf{o}\,b}{\longrightarrow} (\alpha, t')}.$$

These agents, together with agents corresponding to the fork process in (3.8), can then be taken to be the basic components of a calculus with parallel composition, sequential composition and feedback given as in the category of profunctors. This idea is basis for future work.

It remains to explore systematically the full family of models for dataflow, relating automata, event structure and traces-based models to the relational model, following the pattern set in [146].

The higher-order models should be compared to the work in [4].

It would be very interesting to compare the abstract profunctor model to the more concrete models of *scenarios* (Brock and Ackerman [17, 16]). A comparison with the semantics in [123], which unfortunately just recently have come to our attention, should also be carried out.

Early attempts have been made to incorporate fairness into the calculus sketched above and the profunctor model. It is hoped to exploit independence along the lines in [26] and include infinite observations as the work in 7.

Recent work of Selinger [121] takes the first steps in trying to axiomatise the structure of a traced monoidal category in which trace do model a "causally" correct trace, which he refers to as being "loop-like". He gives a characterisation of *uniform* traces as corresponding to loop-like traces, which is able to distinguish the two traces in Rel (cf. Sec. 8.2.1). We hope to be able to show, that his axiomatisation distinguish the two traces in PProf as well.

Part II

Papers

Chapter 4

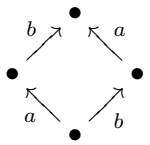
Comparing Transition Systems with Independence and Asynchronous Transition Systems

Abstract: Transition systems with independence and asynchronous transition systems are *noninterleaving* models for concurrency arising from the same simple idea of decorating transitions with events. They differ for the choice of a *derived* versus a *primitive* notion of event which induces considerable differences and makes the two models suitable for different purposes. This opens the problem of investigating their mutual relationships, to which this paper gives a fully comprehensive answer. In details, we characterise the category of *extensional* asynchronous transitions systems as the largest full subcategory of the category of (labelled) asynchronous transition systems which admits TSI, the category of transition systems with independence, as a *coreflective* subcategory. In addition, we introduce *event-maximal* asynchronous transitions systems and we show that their category is *equivalent* to TSI, so providing an exhaustive characterisation of transition systems with independence in terms of asynchronous transition systems.

Introduction

Following the leading idea of CCS [90] and related process calculi [62, 8, 91, 55], the behaviour of concurrent systems is often specified *extensionally* by describing their 'state-transitions' and the observable behaviours that such transitions produce. The simplest formal model of computation able to express naturally this idea is that of *labelled transition systems*, where the labels on the transitions are thought of as the actions of the system at its 'external ports', or, more generally, the observable part of its behaviour.

Transition systems are an *interleaving* model of concurrency, which means that they do not allow to draw a natural distinction between interleaved and concurrent execution of actions. More precisely, transition systems do not model the fact that concurrent actions can overlap in time and reduce concurrency to a nondeterministic choice of action interleavings, so losing track of the casual dependencies between actions and, consequently, of the fact that computations that differ only for the order of independent actions represent, actually, the same behaviour. In other words, interleaving models abstract away from the difference between the factual *temporal* occurrence order and the more conceptual *causal* ordering of actions. The simplest exemplification of this situation is provided by the CCS terms $a \mid b$ and $a.b + b.a$, both described by the following transition system.



(4.1)

Although for many applications this level of abstraction is appropriate, for several other kinds of analysis a model may be desirable that takes full account of concurrency. For instance, apart from any philosophical consideration about the semantic relevance of cause/effect relationships, knowing that different interleavings represent the same behaviour can reduce considerably the state-space explosion problem when checking system properties such as safety properties and fairness [46, 129, 105].

Several efforts have been devoted to the search of transition-based *noninterleaving* models, e.g., transition systems enriched with additional features that make expressing concurrency explicitly possible (cf., e.g., [111, 130, 50, 51, 133, 20]). The present paper focuses on two such models, namely *asynchronous transition systems*, introduced independently by Bednarczyk [11] and Shields [122], and *transitions systems with independence*, proposed by Winskel and Nielsen [146]. These two competing approaches are, among the others, those building on the simplest idea: endow transition systems with some formal notion of 'similarity' of transitions that enables to distinguish whether or not the opposite edges in diagrams such as (4.1) represent the same action. Intuitively, this is achieved in both approaches by thinking of transitions as *occurrences* of *events*, two transitions representing the same event if they correspond to the same action. However, the differences induced on the models by the different choices of how to assign events to transitions are definitely not trivial. And so are the relationships that these models bear to each other.

Getting to the details, asynchronous transition systems assign events to transitions explicitly and enrich the structure further by adding an *independence relation* on the events which describes their causal relationships. This clearly makes distinguishing nondeterminism and concurrency possible; $a.b + b.a$ and $a|b$ can be represented respectively by, e.g., the follow-

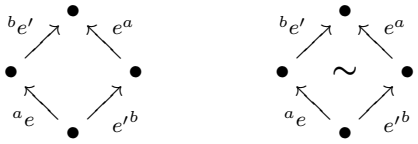


Figure 4.1: Non-determinism vs. independence in labelled asynchronous transition systems

Figure 4.2: Non-determinism vs. independence in transition systems with independence

ing *labelled* asynchronous transition systems, where $\sim$ indicates whether or not the events e and e' (labelled by a and b) are independent.

Observe that here and in the rest of the paper we consider *labelled* asynchronous transition systems [11, 146], i.e., asynchronous transition systems with a further labelling of events, as the proper extension of labelled transition systems.

The expressive power of asynchronous transition systems is clearly not limited to the example above; for instance, Bednarczyk [11] and Mukund and Nielsen [93] have shown that noninterleaving related issues for CCS processes—such as *localities*—can be modelled faithfully using this model. However, it can be argued that assigning both the independence relation and the decoration of transitions with events explicitly means assigning too much. In fact, this obviously introduces some *redundancies* in the model: there are, for instance, many non-isomorphic variations of the asynchronous transitions systems above which can still be reasonably thought as models of $a|b$ and $a.b + b.a$. Moreover, although it is usually easy to tell about independence of transitions, in many important cases it is at least *not* immediate to assign events to transitions: it might very well be the goal of the entire semantic analysis to understand what the events of the system and their mutual relationships are. This consideration seems to indicate that asynchronous transitions systems cannot have a significant impact in Plotkin's SOS style semantics, unless the independence relation is promoted to a greater role.

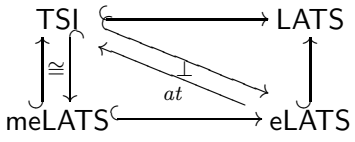
Transition systems with independence are an attempt to answer to the previous observation. Here events are *not* introduced explicitly. They are rather *derived* from the structure of the 'simply-labelled' transitions, upon which the independence relation is directly layered. In such a model, each of the CCS terms discussed above admits only one transition system which can faithfully represent it, viz., respectively, The implicit information about events can be easily deduced from the presence (or the absence) of $\sim$, making the achieved expressive power comparable to that of asynchronous transition systems. Moreover, avoiding a primitive notion of event makes providing a *'noninterleaving'* operational semantics in the SOS style a relatively simple task (cf. [146]).

However, in order to be consistent with the computational intuition, the axiomatics of

transition systems with independence involves (apparently necessarily [118]) *one* condition expressed 'globally' in terms of all the transitions representing occurrences of the same event. This contrasts with the 'local' conditions defining asynchronous transition systems and can make hard checking that a given structure is a transitions system with independence. Thus, the differences induced on the two models by the choice of a *primitive* versus a *derived* notion of event are far-reaching and seem to make them suitable for different applications. This indicates that it is not wise to choose *once and for all* between asynchronous transition systems and transition systems with independence, which, in turn, opens the issue of investigating *formally* their analogies and differences. The contribution of this paper is to answer exhaustively such a question, which, actually, escaped the thorough analysis of models for concurrency carried out in [146, 119, 118]. Precisely, we prove that transition systems with independence besides being nicely related to a class of asynchronous transition systems that we call *extensional*, are *equivalent* to the so-called *event-maximal* asynchronous transition systems. These latter can be seen at the same time as those transition systems that make as few identifications of transitions as possible, i.e., contain no confusion about event identities, and as those in which such identities are derivable from the independence relation, i.e., reduce the redundancy. It is worth mentioning that the converse does not hold: the asynchronous transitions systems for which the independence relation is in turn derivable from the structure of events, and therefore redundant, are slightly less general. They correspond to the transitions systems with independence for which 'independence is concurrency' considered in [119, 118].

Concerning the organization of the paper and its technical contributions, after recalling in Section 4 the definitions of LATS and TSI, respectively the categories of labelled asynchronous transitions systems and of transitions systems with independence, in Section 4.1 we look for a functor adjoint to the obvious embedding $\mathsf{TSI} \hookrightarrow \mathsf{LATS}$. In particular, we identify the category of extensional asynchronous transitions systems, eLATS, as the largest subcategory of LATS which admits TSI as a *coreflective* subcategory. It is worth noticing here that $at\colon \mathsf{eLATS} \to \mathsf{TSI}$, the right adjoint of the coreflection, complements and corrects a non-well-defined construction sketched in [146]: as a matter of fact, due to the greater generality of asynchronous transition systems, eLATS happens to be the largest subcategory of LATS on which such a construction makes sense. Finally, Section 4.2 introduces event-maximal asynchronous transitions systems and their category meLATS, providing the proof of the *equivalence* $\mathsf{TSI} \cong \mathsf{meLATS}$. This yields a complete description of TSI in terms of LATS which can be useful in practise to translate back and forth between the two models when the application one has in mind requires it.

Summing up our results, this paper presents the following commutative diagram, which makes completely formal and precise the relationships between transition systems with independence and asynchronous transition systems.



Preliminaries

In this section we recall briefly the definitions of asynchronous transition systems, transition systems with independence, and their respective categories [11, 146].

As discussed in the introduction, asynchronous transition systems are simply transition systems whose transitions are decorated by events equipped with an independence relation. Four axioms (A1–A4) are needed to guarantee the intended meaning for the events and the independence relation.

Definition 4.0.2 (Labelled Asynchronous Transition Systems) *A labelled asynchronous transition system (lats for short) is a structure*

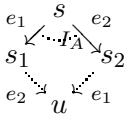
$$A = (S_A, i_A, E_A, \mathit{Tran}_A, I_A, L_A, \ell_A),$$

where $(S_A, i_A, E_A, \mathit{Tran}_A)$ is a transition system with set of states S_A, initial state $i_A \in S_A$, and transitions $\mathit{Tran}_A \subseteq S_A \times E_A \times S_A$, and where E_A is a set of events, L_A a set of labels, $\ell_A \colon E_A \to L_A$ a labelling function, and $I_A \subseteq E_A \times E_A$, the independence relation, is an irreflexive, symmetric relation such that

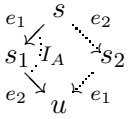
A1. $e \in E_A \quad \Rightarrow \quad \exists s_1, s_2 \in S_A.\ (s_1, e, s_2) \in \mathit{Tran}_A$;

A2. $(s, e, s_1), (s, e, s_2) \in \mathit{Tran}_A \quad \Rightarrow \quad s_1 = s_2$;

A3. $e_1\ I_A\ e_2,\ (s, e_1, s_1), (s, e_2, s_2) \in \mathit{Tran}_A \quad \Rightarrow \quad \exists u.\ (s_1, e_2, u), (s_2, e_1, u) \in \mathit{Tran}_A$;



A4. $e_1\ I_A\ e_2,\ (s, e_1, s_1), (s_1, e_2, u) \in \mathit{Tran}_A \quad \Rightarrow \quad \exists s_2.\ (s, e_2, s_2), (s_2, e_1, u) \in \mathit{Tran}_A$.



In the rest of the paper we shall let $I(e)$ denote the set $\{e' \mid e\ I_A\ e'\}$ and, for convenience, use (s, e^a, s') as a shorthand for a transition (s, e, s') with $\ell_A(e) = a$.

The following is the standard definition of morphisms for lats, which essentially captures the idea of *simulation* (cf. [11, 146]).

Definition 4.0.3 (Asynchronous Transition System Morphisms) *For A and A' lats, a morphism from A to A' is a triple of (partial) functions*[1] *$(\sigma \colon S_A \to S_{A'}, \eta \colon E_A \rightharpoonup E_{A'}, \lambda \colon L_A \rightharpoonup L_{A'})$, where (σ, η) is a morphism of labelled transition systems, i.e.,*

- $\sigma(i_A) = i_{A'}$;
- $(s_1, e, s_2) \in \mathit{Tran}_A,\ \eta(e)\downarrow \quad \Rightarrow \quad \big(\sigma(s_1), \eta(e), \sigma(s_2)\big) \in \mathit{Tran}_{A'}$;
  $(s_1, e, s_2) \in \mathit{Tran}_A,\ \eta(e)\uparrow \quad \Rightarrow \quad \sigma(s_1) = \sigma(s_2)$;

[1] We use, respectively, $f \colon A \to B$ and $f \colon A \rightharpoonup B$ to indicate total and partial functions. For f a partial function, $f(x)\downarrow$ ($f(x)\uparrow$) means that f is (un)defined at x.

which preserves the labelling, i.e., makes the following diagram commutative

$$\begin{array}{ccc} E_A & \xrightarrow{\eta} & E_{A'} \\ {\scriptstyle \ell_A}\downarrow & & \downarrow{\scriptstyle \ell_{A'}} \\ L_A & \xrightarrow[\lambda]{} & L_{A'}; \end{array}$$

and the independence, i.e.,

$$e_1\ I_A\ e_2,\ \eta(e_1)\downarrow,\ \eta(e_2)\downarrow \quad \Rightarrow \quad \eta(e_1)\ I_{A'}\ \eta(e_2).$$

It is immediate to see that lats and their morphisms form a category, which we shall refer as LATS.

Starting from Definition 4.0.2, transition systems with independence attempt to simplify the structure retaining explicitly only the independence, now layered directly on the transitions. As already mentioned, the notion of event becomes implicit, determined by the independence relation through the equivalence $\sim$.

Definition 4.0.4 (Transition Systems with Independence) *A transition system with independence (tsi for short) is a structure*

$$T = (S_T, i_T, L_T, Tran_T, I_T),$$

where $(S_T, i_T, L_T, Tran_T)$ is a transition system and $I_T \subseteq Tran_T \times Tran_T$, the independence relation, is an irreflexive, symmetric relation, such that, denoting by $\prec$ the binary relation on transitions given as

$$\begin{array}{l} (s,a,s_1) \prec (s_2,a,u) \quad \Leftrightarrow \\ \qquad \exists b \in L_T.\ (s,a,s_1)\ I_T\ (s,b,s_2), \\ \qquad\qquad (s,a,s_1)\ I_T\ (s_1,b,u),\ (s,b,s_2)\ I_T\ (s_2,a,u) \end{array}$$

and by $\sim$ the least equivalence on transitions which includes it, we have

T1. $(s,a,s_1) \sim (s,a,s_2) \ \Rightarrow\ s_1 = s_2$;

T2. $(s,a,s_1)\ I_T\ (s,b,s_2) \ \Rightarrow\ \exists u.\ (s,a,s_1)\ I_T\ (s_1,b,u),\ (s,b,s_2)\ I_T\ (s_2,a,u)$;

T3. $(s,a,s_1)\ I_T\ (s_1,b,u) \ \Rightarrow\ \exists s_2.\ (s,a,s_1)\ I_T\ (s,b,s_2),\ (s,b,s_2)\ I_T\ (s_2,a,u)$;

T4. $(s,a,s_1) \prec \cup \succ (s_2,a,u)\ I_T\ (w,b,w') \ \Rightarrow\ (s,a,s_1)\ I_T\ (w,b,w')$.

The $\sim$-equivalence classes, in the following denoted by $[(s,a,s')]$, for (s,a,s') a representative of the class, are to be thought of as events, i.e., $t_1 \prec t_2$ means that t_1 and t_2 are part of a 'concurrency diamond', whilst $t_1 \sim t_2$ means that they are occurrences of the same event. Concerning the axioms, notice then that T1 (the global condition mentioned earlier) corresponds to A2 and axioms T2 and T3 correspond, respectively, to A3 and A4. The role of T4 is to ensure that the independence relation is actually well defined as a relation on events. In the rest of the paper we shall see that this view of $[(s,a,s')]$ agrees with the notion of events for lats and that, in fact, it identifies an interesting subclass of them.

Using $I(t)$ to denote the set $\{t' \mid t\ I_T\ t'\}$, we can state the following lemma which will be useful later on. As a matter of notations, we shall use π_i to denote projections, i.e., if t is (s,a,s'), then $\pi_1(t) = s$, $\pi_2(t) = a$ and $\pi_3(t) = s'$.

Lemma 4.0.5 *Axiom* T4 *is equivalent to*

$$t_1 \sim t_2 \quad \Rightarrow \quad I(t_1) = I(t_2). \tag{T4$'$}$$

Proof. *Easy, by induction.* □

The following definition of morphisms for transition systems with independence resembles closely that given earlier for lats.

Definition 4.0.6 (Transition System with Independence Morphisms) *For T and T' tsi, a morphism from T to T' consists of a pair of (partial) functions $(\sigma\colon S_T \to S_{T'}, \lambda\colon L_T \rightharpoonup L_{T'})$ which is a morphism of transition systems and, in addition, preserves independence, i.e.,*

$$\begin{aligned}(s_1, a, s_2)\ I_T\ (s_1', b, s_2'),\ \lambda(a)\downarrow, \lambda(b)\downarrow \quad &\Rightarrow \\ \big(\sigma(s_1), \lambda(a), \sigma(s_2)\big)\ I_{T'}\ &\big(\sigma(s_1'), \lambda(b), \sigma(s_2')\big).\end{aligned}$$

We shall use TSI to denote the category of tsi and their morphisms.

The following lemma states that tsi morphisms are well defined as maps of events, an easy consequence of the fact that they preserve independence that we shall use in order to embed TSI into LATS.

Lemma 4.0.7 (Morphisms map Events to Events) *For $(\sigma, \lambda)\colon T \to T'$ a morphism of tsi and $(s_1, a, s_2) \sim (s_1', a, s_2')$ equivalent transitions of T, if $\lambda(a)\downarrow$, then $\big(\sigma(s_1), \lambda(a), \sigma(s_2)\big) \sim \big(\sigma(s_1'), \lambda(a), \sigma(s_2')\big)$, i.e., tsi morphisms preserve $\sim$.*

4.1 From LATS to TSI: a coreflection

The scene is now set to expose the adjunction between TSI and a full subcategory of LATS. First, we define an inclusion $ta\colon \mathsf{TSI} \hookrightarrow \mathsf{LATS}$ in the obvious way.

On the objects, ta acts by decorating each transition with the event identified by the $\sim$-class the transition belongs to. The label of such an event is, of course, the label originally carried in the tsi by the transition. Observe that, in force of Definition 4.0.4 of $\sim$, this labelling is well defined. Finally, the independence relation of $ta(T)$ is inherited directly from the one of T. The formal definition is as follows.

Definition 4.1.1 ($\mathsf{TSI} \hookrightarrow \mathsf{LATS}$) *For T a tsi, let $ta(T)$ be the structure $(S_T, i_T, E, Tran, I, L_T, \ell)$, where, denoting by $\sim$ the equivalence relation induced by I_T as in Definition 4.0.4,*

- ▷ *$E = Tran_T/\sim$, the set of $\sim$-classes of $Tran_T$;*
- ▷ *$Tran = \big\{\big(s_1, [(s_1, a, s_2)], s_2\big) \;\big|\; (s_1, a, s_2) \in Tran_T\big\}$;*
- ▷ *$[(s_1, a, s_2)]\ I\ [(s_1', a, s_2')]$ if and only if $(s_1, a, s_2)\ I_T\ (s_1', a, s_2')$;*
- ▷ *$\ell\big([(s_1, a, s_2)]\big) = a$.*

It follows from Lemma 4.0.5 that the definition of the independence on the events of $ta(T)$ is well given. It is now easy to verify the following.

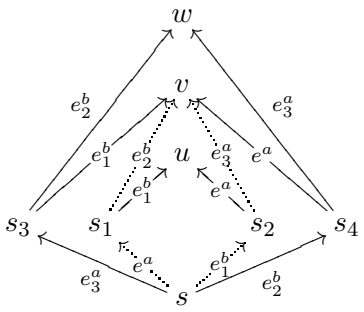


Figure 4.3: A lats that does *not* yield a tsi by abstracting from the events

Proposition 4.1.2 *The transition system $ta(T)$ is a* lats. Proof. *Axiom* A1 *is trivially satisfied. Axiom* A2 *is satisfied because of* T1*, for, by definition of ta, two transitions carry the same event if and only if they belong to the same $\sim$-class in T. Concerning* A3 *and* A4*, they correspond directly to* T2 *and* T3. □

In order to define *ta* as a functor, we need to assign its action on the morphisms in TSI.

Definition 4.1.3 (TSI $\hookrightarrow$ LATS) *For $(\sigma,\lambda)\colon T \to T'$ a morphism of* tsi*, let $ta\big((\sigma,\lambda)\big)$ be (σ,η,λ), where*

$$\eta\big([(s,a,s')]\big) = \begin{cases} \big[(\sigma(s),\lambda(a),\sigma(s'))\big] & \text{if } \lambda(a)\downarrow, \\ \text{undefined} & \text{if } \lambda(a)\uparrow. \end{cases}$$

That Definition 4.1.3 is well given follows from Lemma 4.0.7; it is then easy to check that *ta* is a *full* and *faithful* functor, i.e., an embedding of TSI in LATS.

The obvious idea for a map *at* left inverse to *ta*, as hinted also in [146], is to forget the events and bring the independence from the events down to the transitions, i.e., for A a lats, to take $at(A)$ to be $(S_A, i_A, L_A, Tran, I)$, where

- $(s,a,s') \in Tran$ if and only if $(s,e^a,s') \in Tran_A$,
- $(s,a,s_1)\ I\ (s_2,b,s_3)$ if and only if $(s,e_1^a,s_1),(s_2,e_2^b,s_3) \in Tran_A,\ e_1\ I_A\ e_2$.

This construction, however, contrarily to the claims of [146], is not well defined on the whole LATS, since the interplay between the explicitly given independence and events in lats allows rather complicated situations—of dubious computational significance—which cannot be expressed with tsi. A counterexample is illustrated in Fig. 4.3.
The independent events are $e\ I_A\ e_1$, $e_3\ I_A\ e_1$, $e\ I_A\ e_2$, and $e_2\ I_A\ e_3$, i.e., the system consists of four independency diamonds 'on top of each other'. It is easy to check that this is an object of LATS. However, by applying *at* we create a 'ghost' independency diamond (the one highlighted by the dotted lines), so violating condition T1. In fact, $(s,a,s_3) \sim (s,a,s_1)$ with $s_1 \neq s_3$. This demonstrates that the combination of independence and events makes it hard to define 'uniformly' a map from LATS to TSI to act as left inverse to $ta\colon \mathsf{TSI} \hookrightarrow \mathsf{LATS}$.

However, it is not hard to check that things go smoothly for those lats belonging to the *image* of *ta*. In such a case, *at* lands in TSI and, of course, we have the following result.

Lemma 4.1.4 *For any T in* TSI*, we have $at \circ ta(T) = T$.*

At this point, the issue arises of identifying suitable conditions which, imposed on lats, constrain them down to a category which bears good relationships with TSI. Possibly, one should also like to find a nice characterisation of the image of ta in LATS. We shall do so next, by focusing on *extensional* asynchronous transition systems.

We start by considering lats A satisfying

$$(s_1, e_1^a, s_2) \neq (s_1, e_2^b, s_2) \in Tran_A \quad \Rightarrow \quad a \neq b. \tag{Ex}$$

In words, these are lats where no two transitions between the same states can carry the same label. This is a kind of extensionality condition that, in view of Definition 4.0.4, is clearly necessary for our purposes. In fact, without (Ex), the one-to-one correspondence between morphisms of the kinds $ta(T) \to A$ and $T \to at(A)$—required by the adjointness conditions—would not exist. Next, we let the counterexample discussed above guide us to identify two simple additional conditions—strengthening A3 and A4 with uniqueness criteria—that we shall prove to be *necessary* and *sufficient* in order for at to be well defined on lats satisfying (Ex). As a notation, for $(s, e^a, s') \in Tran_A$, we shall use $at(s, e^a, s')$ to refer to the (unique) transition $(s, a, s') \in Tran_{at(A)}$ it corresponds to.

Proposition 4.1.5 *For A a lats satisfying* (Ex)*, $at(A)$ belongs to* TSI *if and only if*

1. *for $e_1\ I_A\ e_2$ and $(s, e_1^a, s_1), (s, e_2^b, s_2) \in Tran_A$, there exists a unique pair $(s_1, x_2^b, u), (s_2, x_1^a, u) \in Tran_A$ such that $e_1\ I_A\ x_2$, $e_2\ I_A\ x_1$, and $x_1\ I_A\ x_2$.*

2. *for $e_1\ I_A\ e_2$ and $(s, e_1^a, s_1), (s_1, e_2^b, u) \in Tran_A$, there exists a unique pair $(s, x_2^b, s_2), (s_2, x_1^a, u) \in Tran_A$ such that $e_1\ I_A\ x_2$, $e_2\ I_A\ x_1$, and $x_1\ I_A\ x_2$.*

Proof. *The pairs of transitions in i) and ii) exist because of axioms* A3 *and* A4*. If $at(A) \in$* TSI*, their uniqueness is needed in order for $at(A)$ to satisfy axiom* T1*. Suppose that, on the contrary, in case i) there are two pairs $(s_1, x_2^b, u), (s_2, x_1^a, u)$ and $(s_1, y_2^b, w), (s_2, y_1^a, w)$ satisfying the condition. Since A satisfies (*Ex*), we have $w \neq u$, which implies that $at(s, e_2^b, s_2) \prec at(s_1, y_2^b, w)$ and $at(s, e_2^b, s_2) \prec at(s_1, x_2^b, u)$, i.e., that $at(s_1, x_2^b, u) \sim at(s_1, y_2^b, w)$, which contradicts* T1*. The case for ii) can be proved along the same lines, thus showing the necessity of the conditions.*

Concerning their sufficiency, the extensionality guarantees that $I_{at(A)}$ is irreflexive, whilst the property of symmetry for $I_{at(A)}$ is inherited from I_A. It remains check that the axioms T1–T4 *defining* tsi *hold for $at(A)$. Axioms* A3*,* A4 *and conditions i) and ii) above ensure that if $at(t) \prec at(t')$, then $\pi_2(t) = \pi_2(t')$, i.e., t and t' represent the same event. It follows then by induction that $at(t) \sim at(t')$ implies $\pi_2(t) = \pi_2(t')$, for all $at(t), at(t') \in Tran_{at(A)}$. If in addition $\pi_1\big(at(t)\big) = \pi_1\big(at(t')\big)$, then also $\pi_1(t) = \pi_1(t')$ and axiom* A2 *implies that $\pi_3(at(t)) = \pi_3(at(t'))$. So* T1 *is satisfied. Actually, this also implies that* T4 *holds. For, since the independence in $at(A)$ is inherited from that on the events in A, and t and t' carry the same event, we have that $at(t) \sim at(t')$ implies $I(at(t)) = I(at(t'))$. This, as proved by Lemma 4.0.5, is equivalent to* T4*. Finally,* T2 *and* T3 *hold because of the corresponding* A3 *and* A4*.* □

We call *extensional* the lats satisfying (Ex) and the conditions of Proposition 4.1.5, and we denote by eLATS the full subcategory of LATS they determine.

Clearly, at can be extended to a functor from eLATS to TSI which simply 'forgets' the event component of LATS morphisms, i.e., for $(\sigma, \eta, \lambda)\colon A \to A'$, take $at\big((\sigma, \eta, \lambda)\big)$ to be (σ, λ). We shall see next that such a functor is right adjoint to $ta\colon \mathsf{TSI} \hookrightarrow \mathsf{eLATS}$.

Proposition 4.1.6 ($ta \dashv at\colon \mathsf{TSI} \rightharpoonup \mathsf{eLATS}$) *For any $A \in \mathsf{eLATS}$ and any morphism $m\colon T \to at(A)$ in TSI, there exists a unique morphism $m^T\colon ta(T) \to A$ such that $at(m^T) = m$.* Proof. *Let m be (σ, λ). Clearly, by definition of at, m^T must be of the form $(\sigma, \gamma, \lambda)$ for some $\gamma\colon E_{ta(T)} \to E_A$. It is easy to realize that the only possible choice for γ is the following: for $(s, a, s') \in Tran_T$ and $\lambda(a)\downarrow$, let $\gamma([(s, a, s')])$ be the event $e \in E_A$ of the unique transition $\big(\sigma(s), e^{\lambda(a)}, \sigma(s')\big) \in Tran_A$. This is a well given definition, for Lemma 4.0.7 ensures that m maps all transitions in $[(s, a, s')]$ to the same $\sim$-class of $Tran_{at(A)}$, and the proof of Proposition 4.1.5 shows that if two transitions belong to the same $\sim$-class of $Tran_{at(A)}$, they originate from transitions in $Tran_A$ carrying the same event. This proves both existence and uniqueness of m^T.* □

Proposition 4.1.6 proves that the identity natural transformation

$$\eta = \big\{id_T\colon T \to at \circ ta(T)\big\}_{T \in \mathsf{TSI}}$$

is the *unit* of an adjunction involving ta and at. Moreover, since η is an isomorphism, by standard results in category theory, we have that the adjunction $ta \dashv at\colon \mathsf{TSI} \rightharpoonup \mathsf{eLATS}$ is a coreflection, i.e., TSI is *coreflective* in eLATS. This, together with Proposition 4.1.5 and the discussion at the beginning of the present section, shows that eLATS is the largest subcategory of LATS on which at can be defined as a functor to TSI, yielding a right adjoint to ta.

4.2 meLATS: A category of LATS equivalent to TSI

In this section we identify the *replete* image of ta in LATS, i.e., the full subcategory meLATS of eLATS consisting of the objects isomorphic to $ta(T)$, for some $T \in \mathsf{TSI}$. In addition, we characterise those lats for which the independence can be recovered from the structure of events, and relate them to a relevant subcategory of TSI considered in [119, 118].

Recall from basic category theory that meLATS is determined by the coreflection: it consists of those $A \in \mathsf{eLATS}$ for which the corresponding component ϵ_A of the *counit* of $ta \dashv at$ is iso. Applying standard categorical results to derive ϵ from $(-)^T$ and η, we find that it is the natural transformation

$$\epsilon = \big\{(id_{S_A}, \gamma, id_{L_A})\colon ta \circ at(A) \to A\big\}_{A \in \mathsf{eLATS}},$$

where for $(s, a, s') \in Tran_{at(A)}$, $\gamma([(s, a, s')]) =_{def} e$, for $e \in E_A$ the event of the unique $(s, e^a, s') \in Tran_A$. Clearly, ϵ_A is iso if and only if γ is such, i.e.,

$$\forall t, t' \in Tran_A,\ \pi_2(t) = \pi_2(t') \quad \Rightarrow \quad at(t) \sim at(t'),$$

which means that two transitions carry the same event if and only if they belong to the same $\sim$-class of A (viewed as a tsi). Although this characterises $\mathsf{meLATS} \subset \mathsf{LATS}$ equivalent to TSI, it would of course be better to find a more direct description of it, one not referring to $at(A)$. This is the purpose of the notion of *event-maximal* asynchronous transitions systems introduced next.

Intuitively, a lats is *event-maximal* if its events and independence are 'tightly coupled', so that one cannot 'split' events without destroying the global lats structure. More precisely, A is event-maximal if for any $\bar{e} \in E_A$ and any subset T of transitions carrying $\bar{e}$, the structure resulting from replacing $\bar{e}$ on the transitions in T by a *fresh* event $\tilde{e}$ is *no* longer a lats.

Definition 4.2.1 (Event-Maximal Asynchronous Transition Systems) *For A a* LATS*, $\bar{e} \in E_A$, and $T \subset T_{\bar{e}} = \{t \in Tran_A \mid \pi_2(t) = \bar{e}\}$, let $A[T]$ denote the replacement of $\bar{e}$ on the transitions in T for a fresh event $\tilde{e} \notin E_A$, i.e., $A[T] = (S_A, i_A, E_A \cup \{\tilde{e}\}, Tran, I, L_A, \ell)$, where*

- $Tran = \big(Tran_A \setminus T\big) \cup \big\{(s_1, \tilde{e}, s_2) \;\big|\; (s_1, \bar{e}, s_2) \in T\big\}$;
- $I = I_A \cup I_T \cup I_T^{-1}, \quad I_T = \big\{(\tilde{e}, e) \;\big|\; \bar{e}\ I_A\ e\big\}$;
- $\ell(e) = \begin{cases} \ell_A(e) & \text{if } e \in E_A, \\ \ell_A(\bar{e}) & \text{if } e = \tilde{e}. \end{cases}$

A lats A is event-maximal if for each $\bar{e} \in E_A$ and each nonempty $T \subset T_{\bar{e}}$, the transition systems $A[T]$ is not a lats.
The category meLATS *is the full subcategory of* LATS *consisting of the extensional, event-maximal lats.*

Observe that the interesting, nontrivial choices for T are those such that $\varnothing \subset T \subset T_{\bar{e}}$, i.e., those in which at least one $\tilde{e}$-transition is added and at least one $\bar{e}$-transition is kept in $A[T]$. The definition above, stating that any such structure must fail to be a lats, is our way to express that—as remarked in the introduction—the identity of the events in event-maximal lats is forced by the independence relation. This provides us with the direct characterisation of TSI in terms of LATS that we sought.

Proposition 4.2.2 (meLATS $\cong$ TSI) meLATS *is equivalent to* TSI. Proof. *Let A be an extensional lats. We prove that the counit ϵ_A is iso if and only if A belongs to* meLATS*. To this purpose, let γ be the event component of ϵ_A.*

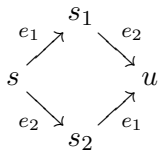
If γ is iso, i.e., for all $t, t' \in Tran_A$ we have that $\pi_2(t) = \pi_2(t')$ implies $at(t) \sim at(t')$, for any choice of $\bar{e} \in E_A$ and any $\varnothing \subset T \subset T_{\bar{e}}$, then the condition in Definition 4.2.1 is satisfied, since, by the extensionality of A, either A3 *or* A4 *must fail for $A[T]$. In fact, in order for $A[T]$ to be a* LATS*, extensionality implies that $t' \in T$ whenever a $at(t') \sim at(t)$ for some $t \in T$, i.e., by the hypothesis on γ, T should be $T_{\bar{e}}$. So A is event-maximal.*

If γ is not iso, i.e., if there exist t and t' such that $at(t) \not\sim at(t')$ but $\pi_2(t) = \pi_2(t')$, then $T = \{t'' \mid at(t'') \sim at(t)\} \subset T_{\pi_2(t)}$ is a nonempty set for which the 'splitting' of $\pi_2(t)$ yields a lats, i.e., A is not event-maximal. □

To conclude this exposition, we observe that the independence relation in event-maximal lats is *not* uniquely determined by rest of the structure. This is due to the fact that the independence on events is still rather *intensional* notion: events may be independent and still never occur in the same path, i.e., intuitively, be mutually exclusive. Observing that such situations have little computational relevance, one may consider on lats the property

$$e_1\ I_A\ e_2 \quad \Rightarrow \quad \exists (s, e_1, s_1), (s, e_2, s_2) \in Tran_A, \tag{E}$$

which can be seen as an extensionality condition on I_A. It is easy to prove that, if $A \in \mathsf{meLATS}$ satisfies (E), then $e_1\ I_A\ e_2$ if and only if there exists a square in A involving e_1 and e_2, i.e.,



Thus, for such lats the independence is completely redundant and can be omitted: all the information is already contained in $(S_A, i_A, E_A, \mathit{Tran}_A, L_A, \ell_A)$.

It is worth remarking here that a condition corresponding to (E) for TSI—viz., whenever $t\ I_T\ t'$, there exist $(s, a, s') \sim t$ and $(s, b, s'') \sim t'$ in Tran_T—was identified in [119, 118] while investigating the tight relationships between tsi and event structures. Such a condition yields TSI_E, a very good-behaved full subcategory of TSI for which we can state the following corollary of Proposition 4.2.2, which concludes the paper. Here we use $\mathsf{meLATS}_\mathsf{E}$ to denote the full subcategory of meLATS consisting of the structures satisfying (E).

Proposition 4.2.3 ($\mathsf{meLATS}_\mathsf{E} \cong \mathsf{TSI}_\mathsf{E}$) $\mathsf{meLATS}_\mathsf{E}$ *is equivalent to* TSI_E.

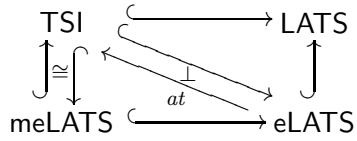
Chapter 5

Transition Systems with Independence and Multi-Arcs

Abstract: We extend the model of transition systems with independence in order to provide it with a feature relevant in the *noninterleaving* analysis of concurrent systems, namely *multi-arcs*. Moreover, we study the relationships between the category of transition systems with independence and multi-arcs and the category of labeled asynchronous transition systems, extending the results recently obtained by the authors for (simple) transition systems with independence (cf. *Proc. CONCUR'96*), and yielding a precise characterisation of transition systems with independence and multi-arcs in terms of (*event-maximal*, *diamond-extensional*) labeled asynchronous transition systems.

Introduction

An exhaustive analysis of the relationship between asynchronous transition systems and transition systems with independence was carried out by the authors in [60], showing that transition systems with independence, besides being nicely related to a class of asynchronous transition systems called *extensional*, are *equivalent* to the so-called *event-maximal* asynchronous transition systems. The results of *loc. cit.* are summarized by the following diagram, where TSI, LATS, eLATS, and meLATS are, respectively, the categories of transitions systems with independence, labeled, extensional, and event-maximal asynchronous transitions systems, and where $\hookrightarrow$, $\perp$, and $\cong$ stand respectively for embeddings, coreflections, and equivalences.



Essentially, the extensionality condition refers to the existence of a *unique* way to 'complete' pairs of independent transitions to '*independence-diamonds*'. Also, it excludes multi-arcs, i.e., multiple transitions with the same label between the same two states. Event-maximality, on the other hand, can be seen at the same time as identifying those transition systems that make as few identifications of transitions as possible, i.e., contain no confusion about event identities, and those in which such identities are derivable from the independence relation, i.e., reduce the redundancy. It is worth noticing here that $at\colon \mathsf{eLATS} \to \mathsf{TSI}$, the right adjoint of the coreflection, complements and corrects a non-well-defined construction sketched in [146]: as a matter of fact, due to the greater generality of asynchronous transition systems, eLATS happens to be the largest subcategory of LATS on which such a construction makes sense.

A question left open by [60] is whether or not the need to restrict to extensional asynchronous transition systems is a consequence of the intrinsic differences between the two notions of events considered, i.e., if in order to be able to model situations ruled out by the extensionality constraints it is necessary to assign events explicitly. This paper addresses such a question; namely, we remove the restriction to transition systems without multi-arcs, relaxing the definition of transition systems with independence, and yielding the new notion of *transition systems with independence and multi-arcs* (*nonextensional transition systems with independence* would probably be a better name, though).

This represents, in our view, an interesting enhancement of the model. In fact, in noninterleaving semantics, to be able to treat multi-arcs is clearly very relevant. In a sense, it can be seen as allowing 'quotienting' of the state-space while retaining full information about events and causality. As an example, consider the CCS term $(a.\mathbf{0}|b.\mathbf{0})+a.b.\mathbf{0}$, traditionally described by the transition system to the left in Fig. 5.1. It is common (see e.g. [91, 93] among others) to *quotient* the state-space by some structural congruence that, e.g., collapses respectively the states b and $\mathbf{0}|b$ and the states $\mathbf{0}|\mathbf{0}$ and $\mathbf{0}$, obtaining the more compact representation — with multi-arcs — shown to the right.

Observe that, contrarily to the interleaving case, it is *vital* here to have *two different* a-transitions (and b-transitions), since they represent different events: one a-transition is part of the independence-diamond and is, therefore, independent of b; the other is not.

In order to justify our definition, we prove that, except for the extensionality condition, the category TSI_m of transition systems with independence and multi-arcs bears exactly the

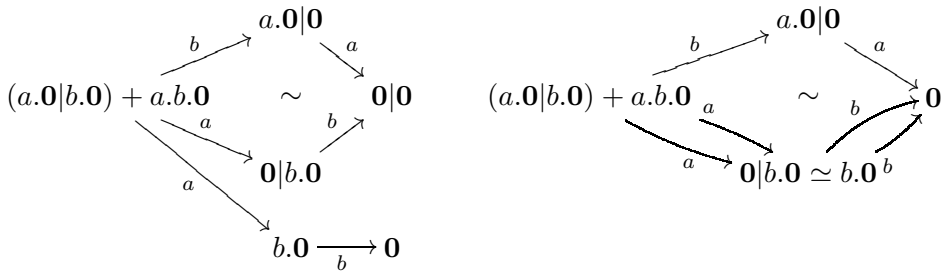
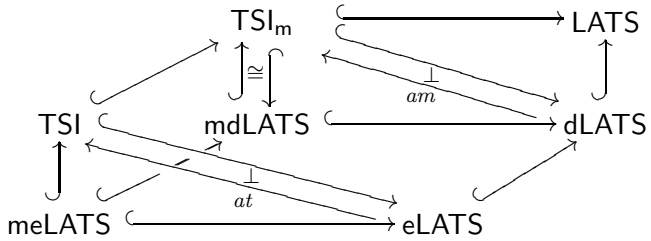


Figure 5.1: In non-interleaving semantics multi-arcs can be essential if the state space is quotiented

same relationships as TSI to LATS. More precisely, we prove that TSI_m is *coreflective* in the category dLATS of the *diamond-extensional* asynchronous transition systems — intuitively, those transition systems that make no confusion about the identities of the events carried by transitions facing each other in independence-diamonds. Similarly to the case of TSI, dLATS is the largest subcategory of LATS for which such a result holds. Moreover, among the *diamond-extensional*, we identify the *event-maximal* asynchronous transition systems and prove that they induce the largest full subcategory of LATS, mdLATS, for which the coreflection cuts down to an *equivalence*. This yields a precise characterisation of TSI_m in terms of LATS that extends the relationships between TSI and LATS discussed above: in fact, the category of eLATS and its full subcategory meLATS are, respectively, the full subcategories of dLATS and mdLATS consisting of transition systems without multi-arcs.

Summing up, this paper presents the following diagram of formal relationships between the new model of transition systems with independence and multi-arcs and asynchronous transition systems which can be useful in practise to translate back and forth between the two models when the application one has in mind requires it.



Although the technical development here goes along the lines of [60], and therefore, strictly speaking, this paper is simply an extension of *loc. cit.*, we believe that the definition of TSI_m is a relevant contribution on its own.

5.1 Comparing LATS with TSI: Considering multi-arcs

In this section we first recall the results of the comparison of TSI and LATS carried out by the authors in [60], and then, reconsidering a restriction used in *loc. cit.*, we introduce the notion of *transition systems with independence and multi-arcs* — i.e., tsi in which multiple transitions carrying the same label are allowed between the same two states. In the next section we shall

then perform an analysis matching that of [60], investigating the relationship between such a category and LATS, and showing that, in a precise sense, our definition provides a minimal, conservative way to extend tsi with multi-arcs.

The starting point of the analysis in [60] is the obvious inclusion $ta\colon \mathsf{TSI} \to \mathsf{LATS}$ which acts on objects by decorating each transition with the event identified by the $\sim$-class the transition belongs to, and by inheriting the independence relation directly from the tsi. On the opposite direction, we considered the 'abstraction' *at* from LATS to TSI that forgets the events and brings the independence from the events down to the transitions. However, a simple argument shows that the presence of multi-arcs in LATS makes it impossible for *at* to be well-defined as a map to TSI. Thus, the very first step of [60] is to consider only those lats A satisfying

$$(\mathsf{Ex}) \qquad (s_1, e_1^a, s_2) \neq (s_1, e_2^b, s_2) \in \mathit{Tran}_A \quad \Rightarrow \quad a \neq b,$$

whose purpose is to forbid multi-arcs. This allows to prove that the *diamond-extensional* asynchronous transition systems, whose definition follows, are exactly those lats A such that $at(A)$ belongs to TSI.

Definition 5.1.1 (Diamond-Extensional lats) *A diamond extensional labeled asynchronous transition system (dlats for short) is a lats that satisfies*

A!3. $e_1\ I_A\ e_2,\ (s, e_1^a, s_1), (s, e_2^b, s_2) \in \mathit{Tran}_A \quad \Rightarrow$
$\exists!\ \mathit{pair}\ (s_1, x_2^b, u), (s_2, x_1^a, u) \in \mathit{Tran}_A.\ e_1\ I_A\ x_2,\ e_2\ I_A\ x_1,\ x_1\ I_A\ \ x_2;$

A!4. $e_1\ I_A\ e_2,\ (s, e_1^a, s_1), (s_1, e_2^b, u) \in \mathit{Tran}_A \quad \Rightarrow$
$\exists!\ \mathit{pair}\ (s, x_2^b, s_2), (s_2, x_1^a, u) \in \mathit{Tran}_A.\ e_1\ I_A\ x_2,\ e_2\ I_A\ x_1,\ x_1\ I_A\ \ x_2.$

The category dLATS *is the full subcategory of* LATS *consisting of the diamond-extensional* lats.

We call *extensional* the diamond-extensional lats that in addition satisfy (Ex), and we denote by eLATS the full subcategory of dLATS that they determine. We can now give the formal definitions of the functors $ta\colon \mathsf{TSI} \to \mathsf{LATS}$ and $at\colon \mathsf{eLATS} \to \mathsf{TSI}$.

Definition 5.1.2 ($\mathsf{TSI} \hookrightarrow \mathsf{LATS}$) *For T a tsi, let $ta(T)$ be the structure*

$$(S_T, i_T, E, \mathit{Tran}, I, L_T, \ell),$$

where, denoting by $\sim$ the equivalence relation induced by I_T as in Definition 4.0.4,

- $E = \mathit{Tran}_T/\!\sim$, *the set of $\sim$-classes of Tran_T;*
- $\mathit{Tran} = \big\{\big(s_1, [(s_1, a, s_2)]_\sim, s_2\big) \bigm| (s_1, a, s_2) \in \mathit{Tran}_T\big\}$;
- $[(s_1, a, s_2)]_\sim\ I\ [(s_1', a, s_2')]_\sim$ *if and only if* $(s_1, a, s_2)\ I_T\ (s_1', a, s_2')$;
- $\ell\big([(s_1, a, s_2)]_\sim\big) = a$.

For $(\sigma, \lambda)\colon T \to T'$ a morphism of tsi, let $ta\big((\sigma, \lambda)\big)$ be (σ, η, λ), where

$$\eta\big([(s, a, s')]_\sim\big) = \begin{cases} \big[(\sigma(s), \lambda(a), \sigma(s'))\big]_\sim & \text{if } \lambda(a)\!\downarrow, \\ \mathit{undefined} & \text{if } \lambda(a)\!\uparrow. \end{cases}$$

The proof that *ta* is well defined follows easily from Lemma 4.0.7. Actually, *ta* is a *full* and *faithful* functor, i.e., an embedding of TSI in LATS. In the following, when no confusion is possible, we may occasionally omit the index $\sim$ from the notation for $\sim$-classes.

Definition 5.1.3 (eLATS $\hookrightarrow$ TSI) *For A a lats, let $at(A)$ be the structure*

$$(S_A, i_A, L_A, \mathit{Tran}, I),$$

where

- $(s, a, s') \in \mathit{Tran}$ *if and only if* $(s, e^a, s') \in \mathit{Tran}_A$,
- $(s, a, s_1)\ I\ (s_2, b, s_3)$ *if and only if* $(s, e_1^a, s_1), (s_2, e_2^b, s_3) \in \mathit{Tran}_A,\ e_1\ I_A\ e_2$.

For $(\sigma, \eta, \lambda)\colon A \to A'$ a morphism of lats, let $at\big((\sigma, \eta, \lambda)\big)$ be (σ, λ).

The result of [60] is that *ta* and *at* form a *coreflection* of TSI in eLATS.

Proposition 5.1.4 ($ta \dashv at\colon \mathsf{TSI} \rightharpoonup \mathsf{eLATS}$) TSI *is coreflective in* eLATS. Proof. *Subsumed by that of the forthcoming Proposition 5.2.8.* ✓

The lats corresponding to tsi are characterised as the *event-maximal* lats. Intuitively, a lats is *event-maximal* if its events and independence are 'tightly coupled', so that one cannot 'split' events without destroying the global lats structure. In other words, the identity of the events in event-maximal lats is forced by the independence relation. This will provide a direct characterisation of tsi in terms of lats

Definition 5.1.5 (Event-Maximal lats) *For A a lats, $\bar{e} \in E_A$, and $T \subset T_{\bar{e}}$, where $T_{\bar{e}} = \{(s, e, s') \in \mathit{Tran}_A \mid e = \bar{e}\}$, let $A[T]$ denote the replacement of $\bar{e}$ on the transitions in T for a fresh event $\tilde{e} \notin E_A$, i.e.,*

$$A[T] = (S_A, i_A, E_A \cup \{\tilde{e}\}, \mathit{Tran}, I, L_A, \ell),$$

where

- $\mathit{Tran} = \big(\mathit{Tran}_A \setminus T\big) \cup \big\{(s_1, \tilde{e}, s_2) \bigm| (s_1, \bar{e}, s_2) \in T\big\}$;
- $I = I_A \cup I_T \cup I_T^{-1}, \quad I_T = \big\{(\tilde{e}, e) \bigm| \bar{e}\ I_A\ e\big\}$;
- $\ell(e) = \begin{cases} \ell_A(e) & \text{if } e \in E_A, \\ \ell_A(\bar{e}) & \text{if } e = \tilde{e}. \end{cases}$

A lats A is event-maximal if for each $\bar{e} \in E_A$ and each nonempty $T \subset T_{\bar{e}}$, the transition systems $A[T]$ is not a lats.
The category mdLATS *is the full subcategory of* LATS *consisting of the diamond-extensional, event-maximal lats.*

The definition above, stating that any structure obtained by 'rearranging' events non-trivially must fail to be a lats, is our way to express that — as remarked before — the identity of the events in event-maximal lats is forced by the independence relation.

Now, if we denote by meLATS the restriction of mdLATS to the full subcategory induced by the objects satisfying (Ex), we can state the final result of [60].

Proposition 5.1.6 (meLATS $\cong$ TSI) meLATS *is equivalent to* TSI. Proof. *Subsumed by that of the forthcoming Proposition 5.2.9.* ✓

Technically, the contribution of this paper is to re-address the choice of condition (Ex) which forbids multiple transitions with the same label between the same two states. Namely, instead of restricting lats in order to get a well-defined functor *at* to TSI, we relax the definition of tsi to allow multi-arcs, proposing below the notion of *transition systems with independence and multi-arcs*. This represents an interesting evolution of tsi, whose relevance goes beyond the comparison of tsi and lats; morally, it constitutes the main contribution of this paper. In other words, we propose here transition systems with independence and multi-arcs and justify their definition by showing how their multi-arcs relates to those of lats.

Formally, we extend tsi in the simplest possible way: transitions are represented by a map assigning to each element of a set *Tran* of transitions a triple consisting of its source, label, and target. This allows to have more transitions between the same two states with the same label simply by having more elements of *Tran* mapped to the same triple. The independence relation and the defining axioms are the obvious translations of those of tsi.

Definition 5.1.7 (tsi with Multi-Arcs) *A transition system with independence and multi-arcs (*$\mathsf{tsi_m}$ *for short) is a structure*

$$T = (S_T, i_T, L_T, \mathit{Tran}_T, \langle - \rangle_T, I_T),$$

where $\langle - \rangle_T \colon \mathit{Tran}_T \to S_T \times L_T \times S_T$ *and* $(S_T, i_T, L_T, \langle \mathit{Tran}_T \rangle_T)$ *is a transition system and* $I_T \subseteq \mathit{Tran}_T \times \mathit{Tran}_T$*, the independence relation, is an irreflexive, symmetric relation, such that, denoting by* $\prec$ *the binary relation on transitions given as*

$$\begin{aligned} t \prec t' \quad \textit{if and only if} \quad & \langle t \rangle_T = (s, a, s_1),\ \langle t' \rangle_T = (s_2, a, u), \\ & \exists t_1, t_2 \in \mathit{Tran}_T.\ \langle t_1 \rangle_T = (s, b, s_2),\ \langle t_2 \rangle_T = (s_1, b, u), \\ & \qquad \textit{with } t\ I_T\ t_1,\ t\ I_T\ t_2,\ t_1\ I_T\ t', \end{aligned}$$

and by $\sim$ *the least equivalence on transitions that includes* $\prec$*, we have*

$\mathsf{T_m}1.\ t \sim t',\ \langle t \rangle_T = (s, a, s_1),\ \langle t' \rangle_T = (s, a, s_2) \quad \Rightarrow \quad t = t';$

$\mathsf{T_m}2.\ t\ I_T\ t',\ \langle t \rangle_T = (s, a, s_1),\ \langle t' \rangle_T = (s, b, s_2) \quad \Rightarrow$
$\exists t_1, t_2.\ \langle t_1 \rangle_T = (s_2, a, u),\ \langle t_2 \rangle_T = (s_1, b, u),\ t\ I_T\ t_2,\ t'\ I_T\ t_1;$

$\mathsf{T_m}3.\ t\ I_T\ t',\ \langle t \rangle_T = (s, a, s_1),\ \langle t' \rangle_T = (s_1, b, u) \quad \Rightarrow$
$\exists t_1, t_2.\ \langle t_1 \rangle_T = (s_2, a, u),\ \langle t_2 \rangle_T = (s, b, s_2),\ t\ I_T\ t_2,\ t_1\ I_T\ t_2;$

$\mathsf{T_m}4.\ t \prec \cup \succ t'\ I_T\ t'' \quad \Rightarrow \quad t\ I_T\ t''.$

As for tsi, the $\sim$-equivalence classes — in the following denoted by $[t]_\sim$, for t a representative of the class — are to be thought of as events. The axioms are recast to fit with the indirect way of assigning source, label, and target to transitions. Notice that a global axiom like $\mathsf{T_m}1$ is still necessary, since the intended notion of events still cannot be determined locally. Axiom $\mathsf{T_m}4$ ensures that the independence relation determines a well-defined relation on events.

In the rest of the paper we shall see that this view of $[t]_\sim$ agrees with the notion of events for lats and that, in fact, $\mathsf{tsi_m}$ relates well to the category of diamond-extensional lats.

Using $I(t)$ to denote the set $\{t' \mid t\ I_T\ t'\}$, we can state the following lemma which will be useful later on. As a matter of notations, we shall use $\langle - \rangle_i$, $i = 1, \ldots, 3$, to denote the composition of $\langle - \rangle_T$ with the appropriate projection, i.e., if $\langle t \rangle_T = (s, a, s')$, then $\langle t \rangle_1 = s$, $\langle t \rangle_2 = a$, and $\langle t \rangle_3 = s'$.

Lemma 5.1.8 *Axiom* $\mathsf{T_m4}$ *is equivalent to*

$$(\mathsf{T^m4'}) \qquad t_1 \sim t_2 \quad \Rightarrow \quad I(t_1) = I(t_2).$$

Proof.
Easy, by induction. ✓

The definition of morphisms for transition systems with independence and multi-arcs necessarily involves a (partial) function on transitions, which, of course, must respect the mapping of states and labels.

Definition 5.1.9 ($\mathsf{tsi_m}$ Morphisms) *For T and T' $\mathsf{tsi_m}$, a morphism from T to T' consists of a triple of (partial) functions*

$$(\sigma\colon S_T \to S_{T'}, \lambda\colon L_T \rightharpoonup L_{T'}, \tau\colon Tran_T \rightharpoonup Tran_{T'})$$

that respects sources, targets, and labels, i.e., that makes the following diagram commute

$$\begin{array}{ccc} Tran_T & \xrightarrow{\quad\tau\quad} & Tran_{T'} \\ {\scriptstyle \langle - \rangle_T}\downarrow & & \downarrow{\scriptstyle \langle - \rangle_{T'}} \\ S_T \times L_T \times S_T & \xrightarrow[<\sigma,\lambda,\sigma>]{} & S_{T'} \times L_{T'} \times S_{T'}, \end{array}$$

preserves independence, i.e.,

$$t\ I_T\ t',\ \tau(t)\!\downarrow,\ \tau(t')\!\downarrow \quad \Rightarrow \quad \tau(t)\ I_{T'}\ \tau(t'),$$

and preserves the 'diamond relation' $\preceq$, i.e.,

$$t \preceq t',\ \tau(t)\!\downarrow\ (or\ \tau(t')\!\downarrow) \quad \Rightarrow \quad \tau(t) \preceq \tau(t').$$

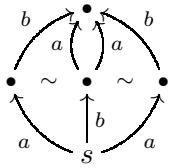
We shall use $\mathsf{TSI_m}$ to denote the category of $\mathsf{tsi_m}$ and their morphisms.

Observe that in the definition above it is necessary to consider the reflexive closure $\preceq$ of the relation $\prec$, since morphisms can be partial and, therefore, collapse diamonds.

Concerning the relationships between TSI and $\mathsf{TSI_m}$, every tsi can be regarded as a $\mathsf{tsi_m}$ simply by defining the map $\langle - \rangle_T$ to act as the 'identity', i.e., interpreting transitions as themselves. Such a mapping extends to an inclusion functor $tm\colon \mathsf{TSI} \hookrightarrow \mathsf{TSI_m}$ by defining $tm\big((\sigma, \lambda)\big)$ to be (σ, λ, τ), where $\tau\big((s, a, s')\big) = (\sigma(s), \lambda(a), \sigma(s'))$. It follows immediately from the last condition in Definition 5.1.9 that τ is well defined as a map of events, a fact that we shall use in later on to embed $\mathsf{TSI_m}$ into LATS.

Lemma 5.1.10 (Morphisms map Events to Events) *For $(\sigma, \lambda, \tau)\colon T \to T'$ a morphism of $\mathsf{tsi_m}$ and $t \sim t'$ equivalent transitions of T, if $\tau(t)\!\downarrow$, then $\tau(t) \sim \tau(t')$, i.e., $\mathsf{tsi_m}$ morphisms preserve $\sim$.*

In general, it is not possible to define a map from TSI_m to TSI that forgets multi-arcs and preserves independence. This is shown by the following example in which collapsing the a-multi-arcs would make the two a-labeled transitions sticking out of s break axiom $\mathsf{T1}$



This means that the embedding $tm\colon \mathsf{TSI} \hookrightarrow \mathsf{TSI}_\mathsf{m}$ does not have a right adjoint. Dually, it can be proved that tm cannot have a left adjont either (a proof that we shall omit, though). Thus, TSI is neither *reflective* nor *coreflective* in TSI_m.

5.2 From LATS to TSI_m: A coreflection

Now that all the bricks are in play, we can complete the picture showing how to extend the functors ta and at to a pair of adjoint functors ma and am forming a coreflection between TSI_m and dLATS. There is only one reasonable way to define the embedding ma.

Definition 5.2.1 ($\mathsf{TSI}_\mathsf{m} \hookrightarrow \mathsf{dLATS}$) *For T a tsi, let $ma(T)$ be the structure*

$$(S_T, i_T, E, \mathit{Tran}, I, L_T, \ell),$$

where, denoting by $\sim$ the equivalence relation induced by I_T as in Definition 5.1.7,

- $E = \mathit{Tran}_T/{\sim}$, *the set of $\sim$-classes of Tran_T;*
- $\mathit{Tran} = \big\{\big(\langle t\rangle_1, [t]_\sim, \langle t\rangle_3\big) \;\big|\; t \in \mathit{Tran}_T\big\}$;
- $[t]_\sim \; I \; [t']_\sim$ *if and only if* $t \; I_T \; t'$;
- $\ell\big([t]_\sim\big) = \langle t\rangle_2$.

It follows from Lemma 4.0.5 and the definition of $\sim$ that the definition of the independence and labels on the events of $ma(T)$ is well given. It is now easy to verify the following.

Proposition 5.2.2 *The transition system $ma(T)$ is a dlats.*
Proof. *Axiom* $\mathsf{A1}$ *is trivially satisfied. Axiom* $\mathsf{A2}$ *is satisfied because of* $\mathsf{T_m1}$*, for, by definition of ma, two transitions carry the same event if and only if they belong to the same $\sim$-class in T. Concerning* $\mathsf{A3}$ *and* $\mathsf{A4}$*, they correspond directly to* $\mathsf{T_m2}$ *and* $\mathsf{T_m3}$*, and the uniqueness criteria imposed by* $\mathsf{A!3}$ *and* $\mathsf{A!4}$ *are a direct consequence of* $\mathsf{T1}$. ✓

In order to define ma as a functor, we need to define its action on the morphisms of TSI_m.

Definition 5.2.3 ($\mathsf{TSI}_\mathsf{m} \hookrightarrow \mathsf{dLATS}$) *For $(\sigma, \lambda, \tau)\colon T \to T'$ a morphism of* $\mathsf{tsi_m}$*, let $ma\big((\sigma, \lambda, \tau)\big)$ be (σ, η, λ), where*

$$\eta\big([t]_\sim\big) = \begin{cases} [t']_\sim & \text{if } \tau(t) = t', \\ \textit{undefined} & \text{if } \tau(t)\uparrow. \end{cases}$$

That Definition 5.2.3 is well given follows from Lemma 5.1.10; it is also easy to check that *ma* is a *full* and *faithful* functor, i.e., an embedding of $\mathsf{TSI_m}$ in dLATS.

The obvious way to define the 'abstraction' *am* to $\mathsf{TSI_m}$ on the objects of LATS is, for a lats A, to make the transitions $Tran_A$ the elements of the transition set $Tran_{am(A)}$ and then interpret them (via $\langle - \rangle_{am(A)}$) simply by replacing the event with its label. We shall prove that this gives a well-defined object-map from the category of diamond-extensional lats to $\mathsf{TSI_m}$, and that dLATS is actually the largest full subcategory of LATS whose every object is mapped by *am* to a $\mathsf{tsi_m}$.

Definition 5.2.4 ($\mathsf{dLATS} \hookrightarrow \mathsf{TSI_m}$) *For A a lats, let $am(A)$ be the structure*

$$(S_A, i_A, L_A, Tran, \langle - \rangle, I),$$

where,

- $(s, e^a, s') \in Tran$ *if and only if* $(s, e^a, s') \in Tran_A$,
- $\langle (s, e^a, s') \rangle = (s, a, s')$,
- $(s, e_1^a, s_1)\ I\ (s_2, e_2^b, s_3)$ *if and only if* $e_1\ I_A\ e_2$.

Proposition 5.2.5 *For A a lats, $am(A)$ belongs to $\mathsf{TSI_m}$ if and only if A belongs to dLATS.*

Proof. *The pairs of transitions in* A!3 *and* A!4 *exist because of axioms* A3 *and* A4. *If $am(A) \in \mathsf{TSI_m}$, their uniqueness is needed in order for $am(A)$ to satisfy axiom* $\mathsf{T_m}1$. *Suppose that, on the contrary, in the case of* A!3 *there are two pairs $(s_1, x_2^b, u), (s_2, x_1^a, u)$ and $(s_1, y_2^b, w), (s_2, y_1^a, w)$ satisfying the condition. Assume, without loss of generality, that $y_2 \neq x_2$. Then we have (s_1, y_2^b, w)*
$\neq (s_1, x_2^b, u)$, but we also have that $(s, e_2^b, s_2) \prec (s_1, y_2^b, w)$ (as transitions of $am(A)$) and $(s, e_2^b, s_2) \prec (s_1, x_2^b, u)$, i.e., that $(s_1, x_2^b, u) \sim (s_1, y_2^b, w)$, which contradicts $\mathsf{T_m}1$. *The case for* A!4 *can be proved along the same lines, thus showing the necessity of the uniqueness conditions.*

Concerning their sufficiency, the property of symmetry and irreflexivity for $I_{am(A)}$ is inherited from I_A. It remains to check that the axioms $\mathsf{T_m}1$–$\mathsf{T_m}4$ *defining* $\mathsf{tsi_m}$ *hold for $am(A)$. Axioms* A3, A4 *and* A!3, A!4 *ensure that if $(s, e_1^a, s_1) \prec (s_2, e_2^a, s_3)$, then $e_1 = e_2$. It follows then by induction that $(s, e_1^a, s_1) \sim (s_2, e_2^a, s_3)$ implies $e_1 = e_2$, for all $(s, e_1^a, s_1), (s_2, e_2^a, s_3) \in Tran_{am(A)}$. If in addition $s = s_2$, then axiom* A2 *implies that $s_1 = s_3$, and so $(s, e_1^a, s_1) = (s_2, e_2^a, s_3)$, i.e.,* $\mathsf{T_m}1$ *is satisfied. Actually, this also implies that* $\mathsf{T_m}4$ *holds. For, since the independence in $am(A)$ is inherited from that on the events in A, we have that $(s, e_1^a, s_1) \sim (s_2, e_2^a, s_3)$ implies $I\big((s, e_1^a, s_1)\big) = I\big((s_2, e_2^a, s_3)\big)$. This, as proved by Lemma 4.0.5, is equivalent to* $\mathsf{T_m}4$. *Finally,* $\mathsf{T_m}2$ *and* $\mathsf{T_m}3$ *hold because of the corresponding* A3 *and* A4. ✓

The definition of *am* on morphisms depends on the fact that LATS morphisms preserve independence on events.

Definition 5.2.6 ($\mathsf{dLATS} \hookrightarrow \mathsf{TSI_m}$) *For $(\sigma, \eta, \lambda)\colon A \to A'$ a morphism of lats, let $am\big((\sigma, \eta, \lambda)\big)$ be (σ, λ, τ), where*

$$\tau\big((s, e^a, s')\big) = \begin{cases} \big(\sigma(s), \eta(e)^{\lambda(a)}, \sigma(s')\big) & \text{if } \eta(e)\downarrow, \\ \text{undefined} & \text{if } \eta(e)\uparrow. \end{cases}$$

By inspecting Definition 5.2.4, it is easy to verify that the above definition makes the diagram in Definition 5.1.9 commute. Moreover, it preserves $\preceq$, since axioms A!3 and A!4 ensure that $e_1 = e_2$, whenever $(s, e_1^a, u) \preceq (s', e_2^a, u')$, i.e., since η preserve independence, $am\big((\sigma, \eta, \lambda)\big)$ is well defined.

In order to prepare for our main proof, we first prove the following lemma.

Lemma 5.2.7 *For any T in $\mathsf{TSI_m}$, we have that $am \circ ma(T)$ is isomorphic to T.*
Proof. *(Sketch) We show that there is a bijection θ between $Tran_T$ and $Tran_{am \circ ma(T)}$ such that (id_S, id_L, θ) and $(id_S, id_L, \theta^{-1})$ are morphisms of $\mathsf{TSI_m}$, respectively from T to $am \circ ma(T)$ and vice versa, inverses of each other. The obvious choice for $\theta(t)$ is $(\langle t \rangle_1, [t]^{\langle t \rangle_2}, \langle t \rangle_3)$. Observe that this gives an injective map because of axiom $\mathsf{T_m}1$.* ✓

The isomorphisms of 5.2.7 directly extends to a natural transformation

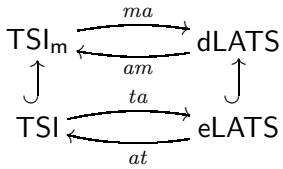
$$\eta = \big\{(id_S, id_L, \theta) \colon T \to am \circ ma(T)\big\}_{T \in \mathsf{TSI_m}} \colon \mathbf{1}_{\mathsf{TSI_m}} \Longrightarrow am \circ ma.$$

We shall prove now that such a transformation is the *unit* of an adjunction involving ma and am, i.e., that am is right adjoint to $ma \colon \mathsf{TSI_m} \hookrightarrow \mathsf{dLATS}$.

Proposition 5.2.8 **($ma \dashv am \colon \mathsf{TSI_m} \rightharpoonup \mathsf{dLATS}$)** *For any $A \in \mathsf{dLATS}$ and any morphism $m \colon T \to am(A)$ in $\mathsf{TSI_m}$, there exists a unique morphism $m^T \colon ma(T) \to A$ in dLATS such that $am(m^T) \circ \eta_T = m$.*
Proof. *Let m be (σ, λ, τ). Clearly, by definition of am, m^T must be of the form $(\sigma, \gamma, \lambda)$ for some $\gamma \colon E_{ma(T)} \to E_A$. It is easy to realize that the only possible choice for γ is the following: for $t \in Tran_T$ and $\tau(t)\downarrow$, let $\gamma([t]) = e$, if $\tau(t) = (s, e^a, s')$. This is a well given definition, for Lemma 5.1.10 ensures that m maps all transitions in $[t]$ to the same $\sim$-class of $Tran_{am(A)}$, and the proof of Proposition 5.2.5 shows that if two transitions belong to the same $\sim$-class of $Tran_{am(A)}$, they originate from transitions in $Tran_A$ carrying the same event. This proves both existence and uniqueness of m^T. Finally, it immediate to check that $am(m^T) \circ \eta_T = m$.* ✓

Since η is an isomorphism, by standard results in category theory, we have that the adjunction $ma \dashv am \colon \mathsf{TSI_m} \rightharpoonup \mathsf{dLATS}$ is a coreflection, i.e., $\mathsf{TSI_m}$ is *coreflective* in dLATS.

Concerning the coreflection described in the previous section, it is immediate to verify that the functors obtained by composing ma and am with the inclusion $tm \colon \mathsf{TSI} \hookrightarrow \mathsf{TSI_m}$ and with the obvious inclusion of eLATS into dLATS coincide, respectively, with ta followed by $\mathsf{eLATS} \hookrightarrow \mathsf{dLATS}$ and with at followed by tm, as illustrated in the following diagram.



This supports our claim of $\mathsf{TSI_m}$ being a *conservative* and *minimal* extension of TSI, since regarding $\mathsf{tsi_m}$ as lats, the extension corresponds exactly to removing the constraint (Ex).

To complete our analysis, we identify the *replete* image of ma in LATS, i.e., the full subcategory mdLATS of dLATS consisting of the objects isomorphic to $ma(T)$, for some $T \in \mathsf{TSI_m}$.

Recall from basic category theory that mdLATS is determined by the coreflection: it consists of those $A \in \mathsf{dLATS}$ for which the corresponding component ϵ_A of the *counit* of $ma \dashv am$ is iso. Applying standard categorical results to derive ϵ from $(-)^T$ and η, we find that it is the natural transformation

$$\epsilon = \left\{(id_{S_A}, \gamma, id_{L_A}) \colon ma \circ am(A) \to A\right\}_{A \in \mathsf{dLATS}} \colon ma \circ am \Longrightarrow \mathbf{1}_{\mathsf{dLATS}},$$

where, for $(s, e^a, s') \in Tran_{am(A)}$, $\gamma([(s, e^a, s')]) = e$. Clearly, ϵ_A is iso if and only if γ is such, i.e.,

$$(s, e_1, s_1), (s_2, e_2, s_3) \in Tran_A,\ e_1 = e_2 \Rightarrow (s, e_1, s_1) \sim (s_2, e_2, s_3) \in Tran_{am(A)},$$

which means that two transitions carry the same event if and only if they belong to the same $\sim$-class of A (viewed as a $\mathsf{tsi_m}$). Expressed purely in terms of LATS, this is, as it was the case for TSI, exactly the event-maximal LATS. Observe that in Definition 5.1.5 the interesting, nontrivial choices for T are those such that $\varnothing \subset T \subset T_{\bar{e}}$, i.e., those in which at least one $\tilde{e}$-transition is added and at least one $\bar{e}$-transition is kept in $A[T]$.

Proposition 5.2.9 ($\mathsf{mdLATS} \cong \mathsf{TSI_m}$) mdLATS *is equivalent to* $\mathsf{TSI_m}$. Proof. *Let A be a diamond-extensional* lats. *We prove that the counit ϵ_A is iso if and only if A belongs to* mdLATS. *To this purpose, let γ be the event component of ϵ_A.*

If γ is iso, i.e., for all $(s, e_1, s_1), (s_2, e_2, s_3) \in Tran_A$ we have that $e_1 = e_2$ implies $(s, e_1, s_1) \sim (s_2, e_2, s_3)$, for any choice of $\bar{e} \in E_A$ and any $\varnothing \subset T \subset T_{\bar{e}}$, then the condition in Definition 5.1.5 is satisfied, since, by the diamond-extensionality of A, either A3 *or* A4 *must fail for $A[T]$. In fact, in order for $A[T]$ to be a* LATS, *diamond-extensionality implies that we must have $(s, e_1, s_1) \in T$ whenever a $(s, e_1, s_1) \sim (s_2, e_2, s_3)$ for some $(s_2, e_2, s_3) \in T$, i.e., by the hypothesis on γ, T should be $T_{\bar{e}}$. So A is event-maximal.*

If γ is not iso, i.e., if there exist (s, e, s_1) and (s_2, e, s_3) such that $(s, e, s_1) \not\sim (s_2, e, s_3)$, then $T = \{(s, e', s') \mid (s, e', s') \sim (s, e, s_1)\} \subset T_e$ is a nonempty set for which the 'splitting' of e yields a lats, *i.e., A is not event-maximal.* ✓

5.3 Conclusion

Based on a comparison between the model of asynchronous transition systems (a model with explicitly defined events) and the model of transition systems with independence (a more abstract model, with a derived notion of events) carried out by the authors in [60], we have introduced the *transition systems with independence and multi-arcs* — a *conservative* and *minimal* extension of transition systems with independence that features multi-arcs — showing that the ability of asynchronous transition systems to model multi-arcs does not depend inherently on the choice of having explicitly given events.

Adding multi-arcs to transition systems with independence constitutes a valuable enhancement to the model, which allows to model important situations in which multiple transitions between the same states represent different events with different causal histories.

Investigating the relationship between the category of transition systems with independence and multi-arcs and the category of labeled asynchronous transition systems that matches the one in [60], we have shown that the former is *coreflective* in the category of *diamond-extensional* labeled asynchronous transition systems, which intuitively are those transition

systems that make no confusion about the identities of the events carried by transitions facing each other in independence-diamonds. This coreflection provides a way to translate semantics forth and back between the two models. Finally, we have identified the *event-maximal* labeled asynchronous transition systems as the largest class of asynchronous transition systems for which the coreflection cuts down to an *equivalence*, so providing a precise characterisation of transition systems with independence and multi-arcs in terms of labeled asynchronous transition systems.

The analysis carried out in this paper helps in deciding when it is necessary to move to a more 'intensional' framework (a lower level of abstraction) in which further distinctions of events are introduced by assigning them explicitly. The definition of transition systems with independence and multi-arcs raises the threshold by allowing a derived notion of event also when multi-arcs are required.

Chapter 6

On Plain and Hereditary History-Preserving Bisimulation

Abstract: We investigate the difference between two well-known notions of independence bisimilarity, *history-preserving bisimulation* and *hereditary history-preserving bisimulation*. We characterise the difference between the two bisimulations in *trace-theoretical* terms, advocating the view that the first is (just) a bisimulation for *causality*, while the second is a bisimulation for *concurrency*. We explore the frontier zone between the two notions by defining a *hierarchy* of bounded backtracking bisimulations. Our goal is to provide a stepping stone for the solution to the intriguing open problem of whether hereditary history-preserving bisimulation is decidable or not. [1] We prove that each of the bounded bisimulations is decidable. However, we also prove that the hierarchy is strict. This rules out the possibility that decidability of the general problem follows directly from the special case. Finally, we give a non trivial reduction solving the general problem for a restricted class of systems and give pointers towards a full answer.

[1] As already remarked in Sec. 3.1, this question has recently been settled in [72].

Introduction

Bisimulation equivalence for concurrent systems was introduced by Park and Milner [104, 87] as a way of describing when two systems can be considered to denote the same process. The idea was to identify systems that could not be distinguished by interaction with an environment, and notably, this took into account the branching structure of systems. It was defined for models for process algebras like e.g. CCS and CSP in which concurrency is treated as *non-deterministic interleaving* of actions. However, for some situations, a more detailed description of the causal ordering between actions is needed. One example is when *action refinement* is considered, as studied by e.g. Vogler [135], Glabbeek and Goltz [131]. Models of this kind, that do not abstract from concurrency, are commonly referred to as *independence*, *partial order* or *true concurrency* models. Examples of these are labelled event structures, Petri nets and asynchronous transition systems, e.g. see [146].

Many attempts have been made to answer the question what the appropriate generalisation of the interleaving bisimulation to independence models is. Two interesting bisimulations for independence models are history-preserving bisimulation (HPB) and hereditary history-preserving bisimulation (HHPB). HPB was introduced in [115] and [32] under the name of *behaviour structure* bisimulation, and *mixed ordering* (mo) bisimulation respectively. The term *history-preserving* originates from [131], where Goltz and vanGlabbeek define the notion for event structures and prove the key property of HPB, namely that it is preserved under action refinement. This result has given history-preserving bisimulation its prominent place among independence bisimulations. In [14] the notion is introduced as fully concurrent bisimulation. There it is independently shown that HPB preserves action refinement for the more general model of Petri nets.

The notion of HHPB first appears in [12], where Bednarczyk studies several history-preserving bisimulations with a downwards closure condition. He calls sets that satisfy this condition *hereditary*. HHPB has also been introduced in [71] under the name of *strong* history-preserving bisimulation. This paper describes a uniform way of defining a bisimulation equivalence across a wide range of different models by applying category-theoretical ideas. For many concrete models, the abstract bisimulation specializes to already known equivalences [28]. In particular, one gets classical bisimulation for standard transition systems. For independence models, the abstract bisimulation specializes to HHPB suggesting that this notion is a very natural independence bisimulation. This is further confirmed by the results of [95]. Relational, logical and game-theoretical characterizations are found which come as conservative extensions of the corresponding characterizations of classical bisimulation.

Altogether a fair amount of work has been done already in studying both, HPB and HHPB. However, very few attempts [132] have been made to demarcate the two notions from each other. Moreover, an intriguing question remains unsolved: Is HHPB decidable for a reasonable class of systems? In contrast, HPB has been shown to be decidable for finite 1-safe Petri nets by Vogler [134], DEXPTIME-complete by Jategaonkar and Meyer [66] and decidable for n-safe nets by Montanari and Pistore [92]. But there is no straightforward adaption of these proofs to HHPB, and it seems that the hereditary condition brings about new dimensions. This justifies a deeper investigation of the difference between plain and hereditary HPB, which is the goal of this paper.

One statement we want to put forward is that hereditary HPB is a bisimulation for *concurrency* as opposed to plain HPB (just) being a bisimulation for *causality*. Intuitively, HPB is an equivalence notion that relates systems with the same *causal* branching structure. It

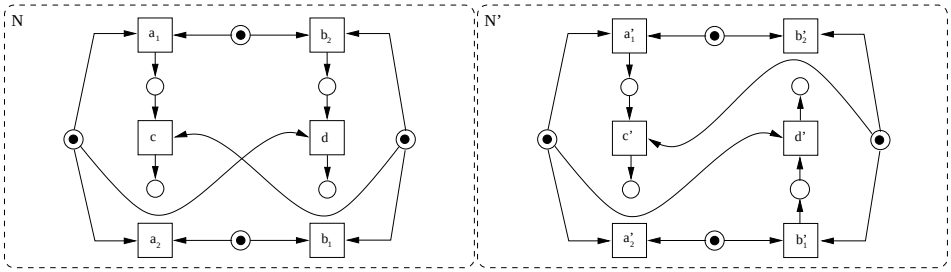


Figure 6.1: Two labelled nets N and N' that are HP bisimilar but not HHP bisimilar. The transitions are labelled by the actions $\{a, b, c, d\}$ as the names suggest, e.g. a_1 is labelled by a

extends the classical notion of bisimulation with the requirement, that any two related runs must have the same causal dependency between actions, that is the same *history*. Hereditary HPB additionally imposes a backtracking condition: for any two related runs, the runs obtained by *backtracking* a pair of related transitions, must be related, too. We allow backtracking not only in the order which is laid down by the related runs; as long as no other transitions depend on a particular transition, it can be backtracked. Thereby it is ensured that the matching is not dependent on the order in which independent actions are linearized. Intuitively this is what we expect from a bisimulation for concurrency.

Figure 6.1 shows the standard example from [95] of two systems that are plain but not hereditary HP bisimilar. Both systems have an a-action (b-action) that can be followed by a dependent c-action (d-action) or an independent (not competing on any places) b-action (a-action). And both have an a-action (a b-action) which can be followed by an independent b-action (a-action). Consequently, the two systems are HP bisimilar. However, observe that in any HPB we can find, the matching of the parallel a- and b-transitions depends on the order in which they appear in the runs to match. So, the systems are not hereditary HP bisimilar. Note that the c transition dictates that we have to match a_1 to a'_1, and so $a_1.b_1$ to $a'_1.b'_1$. Then the backtracking condition requires that b_1 and b'_1 are related. But from this point, the system N' can make a d transition which N cannot match, so b_1 and b'_1 can clearly not be related runs.

After stating the necessary definitions in Sec. 6.1, we present a trace-theoretical characterisation of the difference between HHPB and HPB in Sec. 6.2. This will confirm our view of HHPB as a bisimulation for concurrency as opposed to HPB as a bisimulation for causality. In Sec. 6.3, we consider the effect of restricting HHPB, by bounding how far back in two related runs one can pick transitions to backtrack. Remarkably, we prove in Sec. 6.3.1 that for a fixed bound, each such bisimulation is decidable. However, in Sec. 6.3.2 we find that the bounded bisimulations form a *strict* hierarchy, all trivially stronger than HPB but also strictly weaker than hereditary HPB. In Sec. 6.4 we apply our results to approach the decidability of HHPB (for finite-state systems). After noting that decidability follows almost immediately for the class of bounded asynchronous nets, we present a non-trivial reduction in Sec. 6.4.2 showing that HHPB is decidable for systems with transitive independence relation. In the end, we remark on other partial results and give directions for further progress.

Let us note that one can also consider hidden actions in the context of HPB and HHPB. To avoid confusion with this standard use of strong and weak in the context of bisimulation,

we prefer the name *hereditary* HPB over *strong* HPB. The weak version of HPB has been proved to be decidable in [66] and [136]. Here we will restrict our attention to (hereditary) HPB without hidden actions.

As our model of computation we choose 1-safe Petri nets. However, e.g. by using the results of [146], our results can equally be formulated for other suitable independence models, as for example transition systems with independence or labelled asynchronous transition systems.

6.1 Preliminaries

The following definitions are standard and/or can be found in [66], [94], or [134], perhaps in a slightly varied form.

Petri nets. A labelled Petri net N is a tuple $(S_N, T_N, F_N, init_N, l_N)$, where

- S_N is the set of places,
- T_N is the set of transitions,
- $F_N : (S_N \times T_N) \cup (T_N \times S_N) \to \{0,1\}$ is the flow relation,
- $init_N : S_N \to \mathbb{N}_0$ is the initial marking, and
- $l_N : T_N \to Act$ is the labelling function, where Act is a set of actions.

A net N is *finite* iff S_N and T_N are finite sets.

The pre-set of an element $x \in S_N \cup T_N$, ${}^\bullet x$, is defined by $\{y \mid F_N(y,x) > 0\}$, the post-set of x, $x^\bullet$, similarly is $\{y \mid F_N(x,y) > 0\}$.

A marking M of N is a map $S_N \to \mathbb{N}_0$. We say M enables a transition $t \in T_N$ if $M(s) \geq F(s,t)$ for every $s \in S_N$. If t is enabled at M it can occur. The resulting marking M' is defined by $M'(s) = M(s) - F(s,t) + F(t,s)$ for all $s \in S_N$. We denote this by $M \xrightarrow{t} M'$.

We say that $w = t_1 \dots t_n$, is a *transition-sequence* of N. We write $|w|$ for the length of w, that is $|w| = n$. If $M \xrightarrow{t_1} \cdots \xrightarrow{t_n} M'$ we use $M \xrightarrow{w} M'$ as short notation. For any transition t we write $w.t$ for the sequence $t_1 \dots t_n t$.

A net N is *1-safe* if for every marking M that is reachable from $init_N$, we have: $M(s) \leq 1$ for every $s \in S_N$. Thus, in 1-safe nets a marking can be viewed as a set of places. We say $s \in S_N$ holds at marking M iff $s \in M$. We will always refer to this net class whenever we speak of 'nets' or 'Petri nets' in the following.

Runs. A *run* of a net N is a possibly empty transition-sequence r such that $init_N \xrightarrow{r} M'$ for some M'. Let $Runs(N)$ denote the set of all runs of a net N. When we have $r \in Runs(N)$, $t \in T_N$, and two markings M, M', such that $init_N \xrightarrow{r} M$ and $M \xrightarrow{t} M'$, then we write $r \xrightarrow{t} r.t$.

Independence of Transitions. We say two transitions t and t' of a net N are *independent* in N, denoted by $t \; I_N \; t'$, iff their neighbourhoods of places do not intersect, i. e. iff $({}^\bullet t \cup t^\bullet) \cap ({}^\bullet t' \cup t'^\bullet) = \varnothing$.

Pomsets. A *pomset* is a labelled partial order.[2] It is a tuple $p = (E_p, <_p, L_p, l_p)$, where E_p is a set of events, $<_p$ a partial order relation on E_p, L_p is a set of labels, and l_p a labelling function $l_p : E_p \to L_p$. A function f is an *isomorphism* between pomset p and pomset q iff $f : E_p \to E_q$ is a bijection, such that we have $l_p = l_q \circ f$, and $e <_p e'$ iff $f(e) <_q f(e')$ for all $e, e' \in E_p$.

Transition-pomsets. The *transition-pomset* of a run $r = t_1 \dots t_n$, denoted by $trPom(r)$, has as events the integers from 1 to n, where the label of event i is t_i and the partial ordering is the transitive closure of the following "proximate cause" relation: event i *proximately causes* event j iff $i < j$ and t_i and t_j are *not* independent in N. The *pomset* of r, denoted by $pom(r)$, is the transition-pomset of r, where the label of each event i is $l_N(t_i)$, the *label* of t_i, rather than t_i itself.

Trace Theory. A *trace alphabet* is a pair (Σ, I), where the alphabet Σ is a finite set, and $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric independence relation. Let Σ^* be the set of finite words over Σ, and let r, r' range over Σ^*. For $T \subseteq \Sigma$, let $r \uparrow T$ denote the projection of r onto T, i. e. the sequence obtained by erasing all occurrences of letters which are not in T. The independence relation I induces a relation $\sim_I \subseteq \Sigma^* \times \Sigma^*$ defined by $r \sim_I r'$ iff $r \uparrow \{a, b\} = r' \uparrow \{a, b\}$ for all $a, b \in \Sigma$ such that $\neg(a\ I\ b)$. Clearly, $\sim_I$ is an equivalence relation. The $\sim_I$ equivalence classes are usually referred to as *(Mazurkiewicz's) traces*. For $r \in \Sigma^*$, $[\,r\,]$ stands for the trace containing r. $\Sigma^*/\sim_I$ represents the set of all traces over (Σ, I).

Petri nets and Trace Theory. We can associate the trace alphabet (Σ_N, I_N) to a Petri net N, where $\Sigma_N = T_N$, and I_N is as defined above. Transition-pomsets of a net N correspond one-to-one to traces in $Runs(N)/\sim_{I_N} \subseteq \Sigma_N^*/\sim_{I_N}$. A trace $[r] \in Runs(N)/\sim_{I_N}$ corresponds to $trPom(r)$ and a transition-pomset p of N corresponds to the trace $\{r \mid r \text{ is a linearization of } p\}$.

6.2 (Hereditary) History-Preserving Bisimulation and Trace Theory

We are now ready for the two notions which are central to this paper, *HPB* and *HHPB*. Originally, these bisimulations have been defined on structures that represent the partial order explicitly. By employing the notion of *synchronous runs* from [66], and the notion of *backwards enabled transitions* introduced in [95] we can define (hereditary) HPB on runs, instead. This gives a characterization closely related to work in [32] and [95].

Definition 6.2.1 *Let r_1 and r_2 be runs of nets N_1 and N_2, respectively. We say that r_1 and r_2 are* synchronous *iff the identity function on $\{1, 2, \dots, |r_1|\}$ is an isomorphism between the pomset of r_1 and the pomset of r_2.*

Intuitively, two runs are synchronous if their induced pomsets are isomorphic, and both runs correspond to the same linearization of the associated pomset isomorphism class.

[2] This is not the original definition, but the convention used in [66].

Definition 6.2.2 *Let N be a net, and (Σ_N, I_N) the associated trace alphabet. Let $r = t_1 \dots t_n \in Runs(N)$. For $t \in \Sigma_N$, we say t is* backwards enabled *in r, written $t \in BEn(r)$, iff there is $i \in \{1, \dots, n\}$ s. t. $t_i = t$, and $\forall j \in \{i+1, \dots, n\}$. $t_j \; I_N \; t_i$. This means that i is a maximal element in $pom(r)$. If $t \in BEn(r)$ we define $\delta(r,t)$ to be the result of deleting the last occurrence of t in r, i. e. $\delta(r,t) = t_1 \dots t_{i-1} t_{i+1} \dots t_n$ iff $last(r,t) = i$, where $last(r,t)$ denotes the position of the last occurrence of t in r. That is $last(r,t) = i$ iff $t_i = t$ and $t_j \neq t$ for all $j \in \{i+1, \dots, n\}$.*

Definition 6.2.3 *A* HPB *between two nets N_1 and N_2 consists of a set $\mathcal{H} \subseteq Runs(N_1) \times Runs(N_2)$ of pairs (r_1, r_2) such that*

(i) Whenever $(r_1, r_2) \in \mathcal{H}$, then r_1 and r_2 are synchronous.

(ii) $(\varepsilon, \varepsilon) \in \mathcal{H}$.

(iii) Whenever $(r_1, r_2) \in \mathcal{H}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, then there exists t_2, such that $r_2 \xrightarrow{t_2} r_2.t_2$ and $(r_1.t_1, r_2.t_2) \in \mathcal{H}$.

(iv) Vice versa.

A HPB is hereditary *when it further satisfies*

(v) Whenever $(r_1, r_2) \in \mathcal{H}$ and $t_1 \in BEn(r_1)$ and $t_2 \in BEn(r_2)$ for some t_1, t_2 such that $last(r_1, t_1) = last(r_2, t_2)$, then $(\delta(r_1, t_1), \delta(r_2, t_2)) \in \mathcal{H}$.

We say two nets are (hereditary) HP bisimilar iff there is a (hereditary) HPB relating them.

It is trivial that one can regard a relation $R \subseteq \{ (r_1, r_2) \in T_{N_1}^* \times T_{N_2}^* \mid |r_1| = |r_2| \}$ as a language over the alphabet $T_{N_1} \times T_{N_2}$, and vice versa. With this in mind, we can regard a (hereditary) HPB $\mathcal{H}$ as a language over the *trace alphabet* $\mathcal{T}_{N_1,N_2}$. We define $\mathcal{T}_{N_1,N_2}$ as $\mathcal{T}_{N_1,N_2} = (\Sigma, I)$, where $\Sigma = T_{N_1} \times T_{N_2}$, and I is defined as $(t_1, t_2) \; I \; (t_1', t_2')$ iff $t_1 \; I_{N_1} \; t_1' \; \wedge \; t_2 \; I_{N_2} \; t_2'$.

We will now characterize the difference between HPB and HHPB in trace-theoretical terms. For this we consider two properties of languages.

Definition 6.2.4 *We say a language $L \subseteq \Sigma^*$ is* prefix-closed *iff $r.t \in L$ implies $r \in L$.*

We say L is trace-consistent w. r. t. an independence relation I on Σ *iff $r \sim_I r' \in L$ implies $r \in L$. For $L \subseteq \Sigma^*$, let $L_{\sim_I}$ denote the smallest trace language including L, i. e. $L_{\sim_I} = \{ r \in \Sigma^* \mid \exists r' \in L.\; r' \sim_I r \}$.*

By definition every HHPB is prefix-closed. This does not generally apply for HPBs. But as prefix-closed HPBs correspond to bisimulations that have been built up inductively from $(\varepsilon, \varepsilon)$ without adding "any redundant tuples", we can extract from any given HPB one that is prefix-closed.

Proposition 6.2.5 *Two nets are (hereditary) HP bisimilar iff there exists a* prefix-closed *(hereditary) HPB language relating them.*

A HPB language $\mathcal{H}$ is not necessarily trace-consistent, neither is a HHPB. But this can always be obtained.

Observation 6.2.6 Let $\mathcal{H}$ be a (hereditary) HPB language between two nets N_1 and N_2. Let $\mathcal{T}_{N_1,N_2} = (\Sigma, I)$, then $\mathcal{H}_{\sim_I}$ is a (hereditary) HPB too.

Prop. 6.2.5 ensures, that it is safe to consider only prefix-closed HPBs. Note that if this property is fixed, an analogue to Obs. 6.2.6 is no longer possible. In general, if $\mathcal{H}$ is a prefix-closed HPB, $\mathcal{H}_{\sim_I}$ is not necessarily prefix-closed. However, if $\mathcal{H}$ is hereditary, this will still be true.

Interestingly, if a prefix-closed HPB is also trace-consistent, it is in fact hereditary. So, if one takes as part of the definition that a HPB is prefix-closed, one can regard hereditary HPBs as the class of trace-consistent HPBs.

Proposition 6.2.7 *Two nets are hereditary HP bisimilar iff there exists a* trace-consistent *prefix-closed HPB relating them.*

Proof. "⇒" By Obs. 6.2.6, we can extend every prefix-closed HPB $\mathcal{H}$ to the trace-consistent HPB $\mathcal{H}_{\sim_I}$. If $\mathcal{H}$ is hereditary we have that $\mathcal{H}_{\sim_I}$ is still prefix-closed.

"⇐" Let $\mathcal{H}$ be a trace-consistent and prefix-closed HPB relating the two nets N_1, N_2. We only need to check property (v) of definition 6.2.3. Note that we can use BEn and δ for joint runs and transitions of N_1 and N_2 in the obvious way. Then to prove property (v) we assume $r \in \mathcal{H}$ and $t \in BEn(r)$, and have to show that $\delta(r,t) \in \mathcal{H}$.

So assume $r \in \mathcal{H}$, and $t \in BEn(r)$. As $\mathcal{H}$ is trace-consistent, we have $r' \in \mathcal{H}$ such that r' corresponds to r with the last occurrence of t reshuffled to last position. As $\mathcal{H}$ is prefix-closed, we get $\delta(r',t) = \delta(r,t) \in \mathcal{H}$. □

Remark: Conversely, from Obs. 6.2.6 it follows that one could take as part of the definition that a HPB is trace-consistent. Then HHPBs become the class of prefix-closed HPBs. This is exactly the approach taken in the original definition of HHPB, since HPBs defined on partial orders correspond precisely to the class of trace-consistent HPBs defined on runs. We find the view we have put forward more natural. Taking trace-consistency as part of the definition disguises how the linearized runs of the two systems are matched to each other. Since in HPBs the matching can be dependent on the order in which independent actions are linearized, this is information we do not want to hide away in a HPB.

With the property of prefix-closure we merely restrict our attention to HPBs that have been inductively built up. Hence, defining HPB on synchronous runs and fixing prefix-closure as part of the definition seems very natural. The interpretation of HHPBs as the class of (prefix-closed) HPBs that are *trace languages* expresses then nicely that in HHPB the matching does not depend on the order of how independent transitions are linearized.

It is not difficult to capture the conditions (i)-(iv) of the definition of HPB in terms of languages as well. Together with the results above, this gives a purely language-theoretical characterisation of HPB and HHPB, which can be found in [40].

6.3 History-Preserving Bisimulation and Bounded Backtracking

We define a hierarchy of backtracking bisimulations by bounding the number of transitions which one can backtrack over to an arbitrary number n.

Definition 6.3.1 *A HPB $\mathcal{H}$ is (n)-hereditary when it further satisfies*

(v) Whenever $(r_1, r_2) \in \mathcal{H}$ and $t_1 \in BEn(r_1)$ and $t_2 \in BEn(r_2)$ for some t_1, t_2 such that $last(r_1, t_1) = last(r_2, t_2) \geq |r_1| - n$, then $(\delta(r_1, t_1), \delta(r_2, t_2)) \in \mathcal{H}$.

Note that (0)-hereditary HPBs are exactly the prefix-closed HPBs.

It is easy to give a dynamic condition on nets, which guarantees that (n)-hereditary HP bisimilarity coincides with hereditary HP bisimilarity.

Definition 6.3.2 *Let N be a net. We say that N is* (n)-bounded asynchronous *if for any $r = t_1 t_2 \ldots t_k \in Runs(N)$ such that $t_i \in BEn(r)$, it holds that $k - i \leq n$.*

Proposition 6.3.3 *Let N and N' be two (n)-bounded asynchronous nets. Then N and N' are hereditary HP bisimilar iff N and N' are (n)-hereditary HP bisimilar.*

6.3.1 Decidability of (n)-Hereditary History-Preserving Bisimilarity

For any fixed n, (n)-HHP bisimilarity is decidable for finite systems. The idea behind our proof is that we can define HHPB and (n)-HHPB in a 'forward fashion'. At each tuple we keep a matching directive that prescribes how transitions are going to be matched from this point onwards. The matching directive allows us to express the backtracking requirement as a property of the matching directives of two connected tuples.

To characterize HHPB in this manner we need to record the matching of the entire future. Because of this the forwards characterization merely shifts the difficulty of the decidability of HHPB from the past to the future: now we are confronted with an infinite amount of possible futures. This is not the case for (n)-HHPB. But we shall see that it is sufficient to record future matchings of length n. Our proof builds on this fact and insights gained in the proofs of the decidability of HPB [134, 66].

Below is the definition of (n)-D HPB, our forwards characterization of (n)-HHPB.

Convention. For a pair of synchronous runs (r_1, r_2) of two nets N_1 and N_2, we use r as a short notation. Similarly, we write t for a pair of transitions (t_1, t_2) when t_1 and t_2 correspond to each other in a pair of synchronous runs (r_1, r_2). We also write $r \xrightarrow{t} r'$ when we have two pairs of synchronous runs (r_1, r_2), (r_1', r_2'), and a pair of transitions (t_1, t_2), such that $r_1 \xrightarrow{t_1} r_1'$ and $r_2 \xrightarrow{t_2} r_2'$.

Definition 6.3.4 *A (n)-D HPB between two nets N_1 and N_2 consists of a set $\mathcal{H}_D$ of triples (r_1, r_2, D) such that*

(i) r_1 is a run of N_1, r_2 is a run of N_2, and r_1 and r_2 are synchronous. The matching directive D is a non-empty *and* prefix-closed *set of pairs of words (w_1, w_2), such that w_1 is a transition-sequence of N_1, w_2 of N_2 respectively, and $|w_1| = |w_2| \leq n$.*

(ii) For some D, $(\varepsilon, \varepsilon, D) \in \mathcal{H}_D$.

(iii) Whenever $(r_1, r_2, D) \in \mathcal{H}_D$, and $w \in D$ for some w, such that $|w| < n$, and for some t_1, $r_1.w_1 \xrightarrow{t_1} r_1.w_1.t_1$, then there is some t_2 such that $(w_1.t_1, w_2.t_2) \in D$.
Note that $(\varepsilon, \varepsilon) \in D$ because D is prefix-closed and non-empty.

(iv) Vice versa.

(v) Whenever $(r_1, r_2, D) \in \mathcal{H}_D$*, and* $(t_1, t_2) \in D$*, then there is some* D'*, such that* $(r_1.t_1, r_2.t_2, D') \in \mathcal{H}_D$ *and*

(a) $\forall w$ *s. t.* $|w| < n$. $tw \in D \Leftrightarrow w \in D'$.

(b) $\forall w'$. $w' \in D' \wedge t\ I\ t'$ *for all* $t' \in w' \Rightarrow w' \in D$.

We now prove that (n)-D HPB is indeed equivalent to (n)-HHPB. As in Sec. 6.2 it is sufficient to consider only prefix-closed (n)-D HPBs since they correspond to bisimulations that are built up inductively from the empty runs without adding any "redundant tuples". Prefix-closure for (n)-D HPB is defined as follows.

Definition 6.3.5 *We say a (n)-D HPB* $\mathcal{H}_D$ *is* prefix-closed *iff whenever* $(r_1.t_1, r_2.t_2, D') \in \mathcal{H}_D$*, then there is* $(r_1, r_2, D) \in \mathcal{H}_D$ *for some* D *such that* $t \in D$ *and*

1. $\forall w$ *s. t.* $|w| < n$. $tw \in D \Leftrightarrow w \in D'$.
2. $\forall w'$. $w' \in D' \wedge t\ I\ t'$ *for all* $t' \in w' \Rightarrow w' \in D$.

Lemma 6.3.6 *Two nets are (n)-hereditary HP bisimilar iff they are (n)-D HP bisimilar.*

Proof. For one direction let $\mathcal{H}$ be a (n)-HHPB relating N_1 and N_2. It is also safe to assume prefix-closure of $\mathcal{H}$. We define $\mathcal{H}_D$ by assigning a matching directive D to every pair (r_1, r_2). We take $D = \{w \mid |w| \leq n \wedge r.w \in \mathcal{H}\}$. Prefix-closure of D is given by prefix-closure of $\mathcal{H}$, hence property (i) of definition 6.3.4 clearly holds. Properties (ii), (iii), and (iv) are also trivial.

To see that property (v) holds, let $(r_1, r_2, D) \in \mathcal{H}_D$ and $(t_1, t_2) \in D$. Then, due to the way D is defined there is D' such that $(r.t, D') \in \mathcal{H}_D$. Condition (a) is also immediate by the way matching directives are added to the tuples. To check condition (b) assume we have $w' \in D' \wedge t\ I\ t'$ for all $t' \in w'$. But then we have $r.t.w' \in \mathcal{H}$ with t being backtrack enabled. The fact that $|w'| \leq n$ together with property (v) of definition 6.3.1 implies that $r.w' \in \mathcal{H}$. Hence, by definition of D we have $w' \in D$ as required.

For the other direction assume $\mathcal{H}_D$ to be a prefix-closed (n)-D HPB. Define $\mathcal{H}$ by simply ignoring the matching directive D of triples $(r_1, r_2, D) \in \mathcal{H}_D$. It is clear that properties (i), (ii), (iii) and (iv) of the definition of (n)-HHPB are satisfied. To prove property (v), let $r.t.w \in \mathcal{H}$ such that t is backtrack enabled, and $|w| \leq n$. By prefix-closure of $\mathcal{H}_D$ we have $(r, D), (r.t, D') \in \mathcal{H}_D$ for some D, D' such that $t \in D$, $w \in D'$, and the two conditions of property (v) of definition 6.3.4 are satisfied. But then we have $w \in D$ by condition (b), and thus $(r.w, D'') \in \mathcal{H}_D$ for some D'' as required. □

Now that we have expressed the backtracking condition in a forwards fashion, we can proceed along the lines of the decidability proofs for HPB [134, 66]. We will sketch these proofs, and thereby explain the remaining steps of our decidability proof.

For this we need a further definition from [66].

Definition 6.3.7 *Let* $p = (E_p, <_p, L_p, l_p)$ *be a pomset and* e, $e' \in E_p$*. Event* e' *is a* maximal cause *of event* e *in* p *iff* $e' <_p e$ *and there is no event* $e'' \in E_p$ *such that* $e' <_p e'' <_p e$.

The key insight of the proofs of the decidability of HPB is the following fact: two isomorphic pomsets stay isomorphic after the addition of a pair of transitions iff the maximal causes of the new events are the same (up to isomorphism) in the resulting pomsets. This means that we do not need to keep the entire history, but it is sufficient to record only those events that can act as maximal causes.

The next step is to find a notion that contains this most-recent history, but is finite in the sense that there are only finitely many instances of it. In any partial order run the events that can act as maximal causes correspond to distinct transitions. This is so because a transition cannot be independent of itself. Thus, as one possibility we can take pomsets whose events have distinct transitions as labels. As we consider only finite nets there are clearly only finitely many such pomsets. What we have just described is the notion of growth-sites defined by Jategaonkar and Meyer. Vogler develops a different concept called ordered markings (OM), where the most-recent history is captured by imposing an order on the markings of a net.

Instead of defining HPB on runs we can now base HPB on growth-sites or OMs. The resulting bisimulations are called gsc-bisimulation, and OM-bisimulation, respectively. Jategaonkar and Meyer show that gsc-bisimulation is indeed equivalent to HPB. Vogler proves the analogue for OM-bisimulation. As there are only finitely many growth-sites or OMs for a system, these bisimulations can be decided by exhaustive search. The decidability of HPB is then immediate.

We can define a growth-sites or OM bisimulation that corresponds to (n)-D HPB just as well, and call the resulting notions (n)-D gsc-bisimulation and (n)-D OM-bisimulation. The proof that (n)-D gsc- and (n)-D OM-bisimulation indeed coincide with (n)-D HPB is a straightforward adaptation of the proofs in [66] and [134]. Since there are only finitely many matching directives of size n, (n)-D gsc- and (n)-D OM-bisimilarity can also be decided by exhaustive search. Consequently, (n)-D HP bisimilarity is decidable and with it (n)-HHP bisimilarity.

Theorem 6.3.8 *For any fixed n, it is decidable whether two finite nets are (n)-HHP bisimilar.*

6.3.2 Strictness of the Hierarchy

It is a simple consequence of the definition, that HHP bisimilarity implies (n)-HHP bisimilarity for any n, which again implies (n')-HHP bisimilarity for $n' < n$. Given the result of the previous section, an obvious question to ask is whether HHP bisimilarity coincides with (n)-HHP bisimilarity for some fixed bound n. The example of Fig. 6.1 shows that (0)-HHP bisimilarity is weaker than (1)-HHP bisimilarity. Fig. 6.2 shows an elegant generalisation, which discriminates (n)-hereditary from (n+1)-hereditary HP bisimilarity. Despite its simple appearance, it was not at all trivial to find.

Let us first argue why no HHPB relates N and N'. In any HHPB we must match a_i with a'_i, and b_i with b'_i for $1 \leq i \leq n$. Then one option in N' is to perform a'_{n+1} and b'_{n+1}. These transitions have to be matched with either a_{n+1} and b_{n+1}, or a_{n+2} and b_{n+2} respectively. Suppose we choose the match a_{n+1}, b_{n+1}. We can now backtrack all the a-transitions such that d becomes enabled in N'. But no d action is possible in N. If we choose a_{n+2}, b_{n+2} as our match, we can backtrack all the b-transitions. Then c becomes possible in N', but not in N. The systems are clearly (n+1)-bounded asynchronous, so by Prop. 6.3.3 N and N' are not (n+1)-HHP bisimilar either.

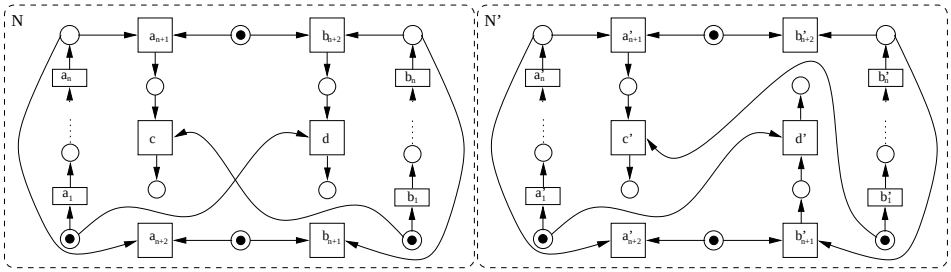


Figure 6.2: Two nets N and N' that are (n)-HHP bisimilar but not (n+1)-HHP bisimilar. Note that for $n = 0$ one gets the two systems given in Fig. 6.1

The above counter-strategy does not apply for (n)-HHPB, but we can use the following strategy to match the critical $n + 1$ transitions. Say we have to match a'_{n+1}, and b'_{n+1} has not been fired yet, i. e. we can still choose between a_{n+1} and a_{n+2} as a match. We make our match dependent on the first transition in the history. Assume it is an a-transition. Then it is safe to match a'_{n+1} with a_{n+1}, which determines that b'_{n+1} is later matched with b_{n+1}. For d to become enabled in N', we need to backtrack all the a-transitions, however there will be $n+1$ b-transitions following the first a, so this is not possible. Similar, it is safe to match a'_{n+2} with a_{n+2}. A symmetrical argument applies if the first action was a b-action, and similar for the remaining cases.

Lemma 6.3.9 *For all $n \in \mathbb{N}_0$, there exist two finite nets that are (n)- but not (n+1)-HHP bisimilar.*

Theorem 6.3.10 *For all $n \in \mathbb{N}_0$, (n)-HHP bisimilarity is strictly weaker than (n+1)-HHP bisimilarity, and hence (unbounded) HHP bisimilarity.*

6.4 Applications to the Decidability Problem of Hereditary History-Preserving Bisimulation

In the previous section we have shown that the hierarchy of (n)-HHPBs is strict. However, for any two fixed finite systems the hierarchy collapses, and so the decidability of the general problem would follow immediately, if the bound can be effectively computed for any two given systems. That this might not be possible in general is indicated by the fact, that the problem of hereditary history-preserving *simulation* has recently been shown to be undecidable [98]. Though, even if the general problem turns out to be undecidable, it is interesting to investigate for which classes of systems deciding HHPB does reduce to deciding (n)-HHPB. Below, we will give some restricted classes of systems, for which this is indeed the case.

6.4.1 Bounded Asynchronous Systems

We say that a net N is bounded asynchronous, if there exists some natural number n such that N is (n)-bounded asynchronous. It is easy to see, that a finite 1-safe net fails to be bounded asynchronous if and only if there is a reachable marking M and a *loop*, $M \xrightarrow{t_1} M_1 \cdots \xrightarrow{t_n} M_n = M$ such that every marking M_i in the loop enables a transition t which is independent of all

transitions in the loop, i.e. $t\ I_N\ t_i$ for all i. Since finite 1-safe nets have only finitely many markings we get the following lemma.

Lemma 6.4.1 *It is decidable if a finite 1-safe net is (n)-bounded asynchronous for some n, and the bound n can be computed.*

With Prop. 6.3.3 the decidability of HHPB for bounded asynchronous systems follows immediately.

Proposition 6.4.2 *HHP bisimilarity is decidable for bounded asynchronous nets.*

6.4.2 Systems with Transitive Independence Relation

Definition 6.4.3 *An independence relation I over an alphabet Σ is transitive if, for every* distinct *t, t', $t'' \in \Sigma$, $t\ I\ t' \wedge t'\ I\ t''$ implies $t\ I\ t''$.*

Let N be a net. A transition $t \in T_N$ is a self-loop *iff ${}^\bullet t = t^\bullet$. Intuitively, a self-loop is a transition that can be repeated immediately, i. e. independently of the occurrence of other transitions. Note that the existence of a run $r = r'.t.t$ implies that t is a self-loop (in our context of 1-safe nets).*

Let us first draw our attention to systems with transitive independence relation that do not contain any self-loops. It is easy to see that for such systems the number of transitions over which can be backtracked is bound by the size of the maximal independence clique. In other words, a system with maximal independence clique of size k is (k)-bounded asynchronous, and hence decidability for finite systems of this subclass is immediate.

If a system contains a self-loop that can occur concurrently with another transition, then this system is clearly not bounded asynchronous. However, we can transfer the decidability result to the full class of finite systems with transitive independence relation with the help of another key observation. In every (H)HPB between two systems with transitive independence relation, concurrently occurring self-loop transitions have always to be matched to self-loops. Hence, we do not need to consider the unfoldings of such self-loops. It is sufficient to match the first occurrence of such a transition, when we make sure that the match is indeed a self-loop. But then the number of transitions over which one can backtrack is again bound by the size of the maximal independence clique, and so we have established decidability. The precise definition of what it means for a self-loop to occur concurrently in a given context, and the details of the proof can be found in the appendix.

Theorem 6.4.4 *For finite systems with transitive independence relation, HHP bisimilarity is decidable.*

6.5 Final Remarks

There is still undiscovered land in the zone between plain and hereditary HPB. One possibility to advance the frontier is to identify system classes for which the two notions coincide. Several classes of such systems have already been found. The most interesting one is the system class of BPP in full standard form [40]. Plain and hereditary HPB for the class of *free-choice* nets have recently been shown not to coincide by the first author, disproving a conjecture in [27].

The trace-theoretical characterization looks promising for approaching the decidability problem of HHPB, see [40] for more details.

Acknowledgements. The first author would like to thank Julian Bradfield and Anca Muscholl for helpful discussions and Walter Vogler for valuable comments on an earlier draft of this paper. The second author would like to acknowledge the lot of people who have contributed with their view on this problem, in particular Marcin Jurdziński.

6.6 Appendix: Proofs for Section 6.4.2

Here we will give the detailed proof of the decidability of HHPB for the full class of finite systems with transitive independence relation. As described in Sec. 6.4.2 the essence of the proof is the observation that concurrently occurring self-loops have always to be matched to self-loops. We will first give the precise definition of what it means for a self-loop to occur concurrently, and then formulate and prove the corresponding lemma.

Definition 6.6.1 *Assume a given net N. Let t be a self-loop transition of N, and let r be some run of N. We say the self-loop t is* concurrently occurring at r *iff*

- *t is enabled at r, and*
- *there exists t', s. t. $t\ I\ t'$ and we have $r \xrightarrow{t'} r.t'$ or $BEn(r, t')$.*

Lemma 6.6.2 *Let $\mathcal{H}$ be a history-preserving bisimulation relating two nets with transitive independence relation, N_1, N_2.*

- *Whenever $(r_1.t_1, r_2.t_2) \in \mathcal{H}$, and t_1 is a concurrently occurring self-loop at r_1, then t_2 is a self-loop as well.*
- *Vice versa.*

Proof. To prove the first part of the lemma let $(r_1.t_1, r_2.t_2) \in \mathcal{H}$ and let t_1 be a concurrently occurring self-loop at r_1. First assume we have $t_1'\ I\ t_1$, such that $r_1 \xrightarrow{t_1'} r_1.t_1'$. Clearly we have $(r_1.t_1.t_1, r_2.t_2.t_2^*) \in \mathcal{H}$ for some $t_2^*\ D\ t_2$, and $(r_1.t_1.t_1.t_1', r_2.t_2.t_2^*.t_2') \in \mathcal{H}$ for some t_2', s. t. $t_2'\ I\ t_2$ and $t_2'\ I\ t_2^*$. With transitivity of independence the latter leads to a contradiction with the requirement $t_2^*\ D\ t_2$, unless $t_2^* = t_2$. But if $t_2^* = t_2$, then t_2 must be a self-loop because it can occur twice consecutively.

Secondly, assume we have $t_1'\ I\ t_1$, such that $BEn(r_1, t_1')$. A similar argument shows that t_2 must be a self-loop, too.

The second part of the lemma can be proved by a symmetric argument. □

This lemma ensures that we do not need to consider the unfoldings of concurrently occurring self-loops. It is sufficient to match one instance of a concurrently occurring self-loop transition, and to make sure it is really matched to a self-loop.

This idea is translated into what we shall call 'No Self-loop Unfolding' (NSU) HPB. After giving the definition we will show that for systems with transitive independence relation this new kind of bisimilarity indeed coincides with (hereditary) history-preserving bisimilarity.

Note that in the following we will make use of the convention introduced in Sec. 6.3.1.

Definition 6.6.3 *A NSU (No Self-loop Unfolding) history-preserving bisimulation between two nets N_1 and N_2 consists of a set $\mathcal{H}_{NSU}$ of pairs (r_1, r_2) such that*

(i) Whenever $(r_1, r_2) \in \mathcal{H}_{NSU}$, then r_1 is a run of N_1, r_2 is a run of N_2, and r_1 and r_2 are synchronous.

(ii) $(\varepsilon, \varepsilon) \in \mathcal{H}_{NSU}$.

(iii) Whenever $(r_1, r_2) \in \mathcal{H}_{NSU}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, such that t_1 is not *a concurrently occurring self-loop at r_1, then there exists t_2, such that $r_2 \xrightarrow{t_2} r_2.t_2$ and $(r_1.t_1, r_2.t_2) \in \mathcal{H}_{NSU}$.*

(iv) Vice versa.

(v) Whenever $(r_1, r_2) \in \mathcal{H}_{NSU}$ and $r_1 \xrightarrow{t_1} r_1.t_1$ for some t_1, such that t_1 is a concurrently occurring self-loop at r_1, and there exists no x_2 such that $(t_1, x_2) \in BEn(r)$, then there exists t_2, such that t_2 is a self-loop, $r_2 \xrightarrow{t_2} r_2.t_2$, and $(r_1.t_1, r_2.t_2) \in \mathcal{H}_{NSU}$.

(vi) Vice versa.

A NSU history-preserving bisimulation is hereditary when it further satisfies

(vii) Whenever $(r_1, r_2) \in \mathcal{H}_{NSU}$ and $t_1 \in BEn(r_1)$ and $t_2 \in BEn(r_2)$ for some t_1, t_2 such that $last(r_1, t_1) = last(r_2, t_2)$, then $(\delta(r_1, t_1), \delta(r_2, t_2)) \in \mathcal{H}_{NSU}$.

We say two nets are (hereditary) NSU history-preserving bisimilar iff there is a (hereditary) NSU HPB relating them.

Lemma 6.6.4 *Two nets with transitive independence relation are (hereditary) history-preserving bisimilar iff they are (hereditary) NSU history-preserving bisimilar.*

Proof. With lemma 6.6.2 it is easy to check that every (hereditary) HPB is also a (hereditary) NSU HPB.

For the non-trivial direction let $\mathcal{H}_{NSU}$ be a (hereditary) NSU HPB. Define $\mathcal{H}$ by unfolding self-loop matches inductively as follows:

Base Step $\mathcal{H} = \mathcal{H}_{NSU}$,

Inductive Step Whenever $rr' \in \mathcal{H}$ and t_1, t_2 is a pair of concurrently occurring self-loops at r_1, r_2, s. t. $(t_1, t_2) \in BEn(r)$ then $r.t.r' \in \mathcal{H}$.

It is easy to check that $\mathcal{H}$ is a (hereditary) HPB. □

We can restrict our attention to the special class of *minimal* (hereditary) NSU HPBs, which strictly do not contain any unfoldings of concurrently occurring self-loops.

Definition 6.6.5 *A (hereditary) NSU HPB $\mathcal{H}_{NSU}$ is* minimal *iff*

- *Whenever $r.t.r' \in \mathcal{H}_{NSU}$ and t_1 is a concurrently occurring self-loop at r_1, then there exists* no *x_2 such that $(t_1, x_2) \in BEn(r)$.*
- *Vice versa.*

Lemma 6.6.6 *Two nets are (hereditary) NSU history-preserving bisimilar iff there exists a minimal (hereditary) NSU history-preserving bisimulation.*

Proof. We can simply 'collapse' any given (hereditary) NSU HPB $\mathcal{H}_{NSU}$ to a minimal one: erase all tuples that violate the above conditions from $\mathcal{H}_{NSU}$. Clearly, the result is still a (hereditary) NSU HPB. □

Minimal (hereditary) NSU HPBs between systems of our subclass look exactly like (hereditary) HPBs of systems with transitive independence relation and no self-loops. They meet all characterisics that made it possible to find a decision procedure for the latter subclass. In particular, the number of joint transitions which one can backtrack over is bound by the size of the maximal independence clique. So, we get the following result.

Lemma 6.6.7 *Hereditary NSU HP bisimilarity is decidable for finite systems with transitive independence relation.*

Proof. By lemma 6.6.6 it is sufficient to check whether there exists a minimal hereditary NSU history-preserving bisimulation. But this is clearly decidable for our subclass. We only need to adapt the steps of the proof of the decidability of (n)-hereditary HPB to show that the corresponding notion of (n)-hereditary NSU HPB is decidable for our subclass. □

With this and lemma 6.6.4 we immediately get decidability for the whole class of finite systems with transitive independence relation.

Chapter 7

A Fully Abstract Presheaf Semantics of SCCS with Finite Delay

Abstract: We present a presheaf model for the observation of *infinite* as well as finite computations. We apply it to give a *denotational* semantics of SCCS with finite delay, in which the meanings of recursion are given by *final* coalgebras and meanings of finite delay by *initial* algebras of the process equations for delay. This can be viewed as a first step in representing *fairness* in presheaf semantics. We give a concrete representation of the presheaf model as a category of *generalised synchronisation trees* and show that it is coreflective in a category of *generalised transition systems*, which are a special case of the general transition systems of Hennessy and Stirling. The open map bisimulation is shown to coincide with the *extended bisimulation* of Hennessy and Stirling. Finally we formulate Milners operational semantics of SCCS with finite delay in terms of generalised transition systems and prove that the presheaf semantics is *fully abstract* with respect to extended bisimulation.

Introduction

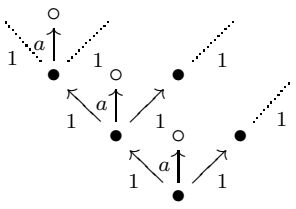
When reasoning about and describing the behaviour of concurrent agents it is often the case that some infinite computations are considered *unfair* and consequently ruled out as being *inadmissible*. An economical way of studying this situation was proposed by Milner in [89] showing how to express a fair parallel composition in his calculus SCCS (*synchronous* CCS) by adding a *finite, but unbounded* delay operator. Syntactically the finite delay of an agent t is written ϵt. The agent ϵt can perform an unbounded number of 1-actions $\epsilon t \xrightarrow{1} \epsilon t$ (delays) but *must eventually* perform an action $\epsilon t \xrightarrow{a} t'$ if t can perform an action $t \xrightarrow{a} t'$ or *stop* if t cannot perform any actions. In other words, its actions are the same as for (the possibly infinite delay) $\delta t = \mathsf{rec}\,x.(1 : x + t)$, except that infinite unfolding of the recursion is not allowed.

To deal with agents in which only some infinite computations are admissible, one must readdress the issue of how to represent the behaviour of agents and so when two agents behave equally, i.e. they denote the same process. The approach used for CCS and SCCS, taking two agents to be equivalent if their derivation trees are strong bisimilar [87], will identify agents that only differ on whether some infinite computations are admissible or not, in particular ϵt is identified with δt for any term t. Moreover, (by definition) both ϵt and δt should be solutions to the equation

$$x \cong (1 : x + t) \tag{7.1}$$

(up to equivalence) so process equations will not have unique solutions as it is the case in CCS and SCCS (with guarded recursion).

In [89], Milner proposes a behavioural preorder called *fortification*, which is designed such that (1) it induces an equivalence which distinguishes the two notions of delay and coincides with strong bisimulation for "standard" agents, (2) recursive processes are *least* fixed points of the associated process equations and (3) the equivalence is a congruence with respect to all the operators of the language (under an assumption of guarded recursion). This approach works reasonably, but is not completely satisfactory. As pointed out by Aczel in [7], the fortification equivalence makes some non desirable identifications of agents due to the fact that infinite computations are treated totally separately from finite computations. For example, the two agents $\delta(a:0 + \delta 0)$ and $\epsilon(a:0 + \delta 0)$ (where 0 is the agent without any actions) are identified by the fortification equivalence. Both agents get assigned the derivation tree



, (7.2)

for which the admissible infinite action sequences of the agents underlying the nodes are the same: For a black node, the underlying agent is either the original agent or the agent $\delta 0$, for which 1^ω is the *only* admissible infinite action sequence. The underlying agent of a white node is the agent 0, which has no action sequences at all. So, the isomorphism between the derivation trees of the two agents is a bisimulation satisfying that the underlying agents of any two related nodes have the same *set* of admissible infinite action sequences. This implies the

fortification equivalence. However, for a true branching equivalence, the two agents should not be equivalent. The first agent can delay infinitely *remaining* able to perform an a-action at any time, while the second agent must reach a state in which it cannot perform an a-action. Aczel [7] proposes a final-coalgebra semantics, which gives a bisimulation closely related to the *extended* bisimulation introduced by Hennessy and Stirling in [54] for *general transition systems*. This bisimulation indeed distinguishes the two agents given above.

The background of the present paper is the work on presenting models for concurrency categorically as initiated by Winskel and Nielsen [146] and developed further in the work on bisimulation from open maps [71] and presheaf models for concurrency [24, 22, 59, 144]. Our goal is twofold: We want to extend the categorical approach (in which the issue of infinite computations and fairness has been absent so far) to models for infinite computations and we want to give a denotational semantics to SCCS with finite delay which captures a behavioural equivalence similar to the extended bisimulation of [54]. As we will see, these two goals can indeed be met.

One of the forces of describing models for concurrency within the language of category theory is that different models suitable for different purposes, can be formally related to each other. E.g. in [146] the category of synchronisation trees suitable for giving denotational semantics to CCS-like process calculi is shown to be a coreflective subcategory of the category of transition systems suited for operational semantics. Another force was added by the notion of *bisimulation from open maps* introduced in [71], from which one gets an abstract behavioural equivalence by choosing a *path category*, i.e. a subcategory of the model at issue identifying the *observable computations*. The open maps approach increased its worth through the further development [24, 22, 21, 144] of the *presheaf* models for concurrency proposed in [71]. Here one *starts* with a path category P and then takes the category $\widehat{\mathsf{P}}$ of presheaves over P as model, justified categorically by being the free colimit completion of P [22]. Now any presheaf model $\widehat{\mathsf{P}}$ comes with a *canonical* notion of bisimulation, taking P as the path category. In [22, 144, 24] it is shown that presheaf models themselves can be related within a category in which arrows are (connected) colimit preserving functors. Such functors preserve the canonical bisimulation and general techniques for their construction are provided.

Perhaps the simplest example of a presheaf model is obtained from the category Fin of all finite sequences of actions from a set Act ordered by the usual prefix ordering. The category $\widehat{\mathsf{Fin}}$ is equivalent to the category of (Act) labelled synchronisation trees and the typical constructions of a CCS-like language can be expressed as functors preserving the canonical equivalence [71, 22]. In this light, it was natural to approach a generalisation of the categorical models to models for infinite computations by studying the presheaf category $\widehat{\mathsf{Inf}}$, where $\mathsf{Inf} = \mathsf{Fin} \cup Act^{\omega}$ is the path category obtained by adding all *infinite* sequences of actions to the category Fin. With the help of a simple *Grothendieck topology* we get indeed a suitable model for infinite computations from the category of *separated presheaves* [83] over Inf. A careful generalisation of the models of synchronisation trees and transition systems lifts the relationship between the "standard" finitary models to the infinitary models and gives a concrete representation of the presheaf model for infinite computations as generalised synchronisation trees, coreflective in a category of generalised transition systems. The generalised transition systems are defined as instances of the general transition systems of [54] and it turns out that the extended bisimulation defined in [54] coincides with the abstract bisimulation obtained from open maps. We show how to give an operational semantics of SCCS with finite delay in the generalised transition systems capturing exactly the definition of inadmissible com-

putations given in terms of waiting subcomputations in [88]. We then give a denotational semantics in the presheaf model which we prove to be *equationally fully abstract* with respect to *extended* bisimulation.

In all of the steps above we greatly benefit from the categorical presentation. *Unbounded non-determinism* is represented simply by (infinite) coproducts. By utilizing the general techniques from [24] we get very simple definitions of the denotations for prefixing and synchronous product, for which congruence properties follow almost for free. As meanings of recursion we take *final coalgebras*, corresponding to *greatest* fixed points and the finite delay operator is simply obtained as an initial algebra corresponding to a *least* fixed point of the process equation (7.1) given above. Finally, the categorical relationships between the different models and the general theory of bisimulation from open maps reduce the problem of relating the two semantics to finding an open map within the category of generalised transition systems.

A number of papers [7, 65, 53, 54, 139] have already proposed denotational semantics for SCCS with finite delay and models for non-deterministic processes with infinite computations. As mentioned above, the approach we take is closely related to the work in [7] and [54]. However, the admissible infinite computations in [7] appear to be identified in a rather syntax dependent way as opposed to simply arising from the use of final coalgebras in giving meanings to recursion. The semantics given in [65] is also shown to be fully abstract, but with respect to the *fortification* equivalence, so it makes the non-intuitive identifications described above. Moreover, it only covers *bounded non-determinism* as obtained from terms in which only a binary sum is allowed. The semantics given in [53] focuses on the fortification equivalence too. Also, for all the models given in [65, 53, 139] the order relation between elements is designed such that meanings of recursion can be given by *least* fixed points using a *reverse* ordering on infinite observations.

The structure of the paper is as follows. In Sec. 7.1 we give some preliminary definitions and recall the categorical concepts used in the paper. In Sec. 7.2 we recall the calculus SCCS [89], the finite delay operator and how to derive a fair parallel [88]. In Sec. 7.3 we introduce respectively the new presheaf model and the transition system models for infinite computations. Section 7.4 is devoted to the bisimulation obtained from open maps and its relationship to the extended bisimulation of [54]. In Sec. 7.5 we formulate Milner's operational semantics of SCCS with finite delay in terms of the generalised transition systems introduced in Sec. 7.3 and in Sec. 7.6 we give the presheaf semantics and the full abstraction result. Comments on future work is given in Sec. 7.7. The appendixes contain details on Grothendieck topologies and the proof of full abstraction.

7.1 Preliminaries

Notation 7.1.1 For a set S, let S^* denote the set of finite, (possibly empty) sequences and S^+ the set of finite non-empty sequences. Let S^ω denote the set of infinite sequences and define $S^\infty = S^+ \cup S^\omega$, i.e. the set of non-empty finite or infinite sequences. We will let Roman letters range over elements and Greek letters range over sequences. Let $|\alpha|$ denote the length of α. If $j \in \omega$ and $|\alpha| \geq j+1$, let $\alpha(j) = \alpha_0\alpha_1 \ldots \alpha_j$, i.e. the first j actions of α. For α, α' such that $|\alpha| < \omega$ we write $\alpha\alpha'$ for the composition of the two sequences. If $\beta \in S^*$ and $\beta \leq \alpha$ in $S^* \cup S^\omega$, we will write $\beta \leq_f \alpha$ for β is *finite and below* α.

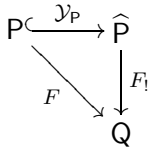
Assume a fixed set *Act* of actions. We will consider finite or possibly infinite sequences of

actions from Act ordered by the standard prefix order. In particular we will let Fin and Inf refer to the two partial order categories Act^+ and Act^∞ (i.e. $Act^+ \cup Act^\omega$) obtained in this way. They will play the key role as *path categories* of presheaf models for the observation of respectively finite and possibly infinite computations.

7.1.1 Presheaf Models, Bisimulation from Open Maps and Transition Systems

Presheaf categories were suggested in [71] as abstract models for concurrency, equipped with a canonical notion of bisimulation equivalence.

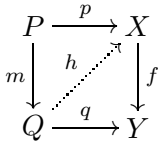
The basic idea is to start from a (partial order) category P defining the *observable computations* or *path shapes* of interest. The category $\widehat{\mathsf{P}}$ of *presheaves over* P is then taken as the category of processes with such path shapes. The category $\widehat{\mathsf{P}}$ is the *free colimit completion* of P, i.e. the category obtained (up to equivalence) by freely adding all colimits to P. It has as objects all functors $X\colon \mathsf{P}^{\mathsf{op}} \to \mathsf{Set}$ (where Set is the category of all small sets and functions between them) and as arrows natural transformations between such. Any functor $F\colon \mathsf{P} \to \mathsf{Q}$ for Q a cocomplete category (i.e. a category having all colimits), can be extended freely (as a left Kan extension [82]) to a (colimit preserving) functor $F_!\colon \widehat{\mathsf{P}} \to \mathsf{Q}$ making the diagram



commute. The functor $\mathcal{Y}_\mathsf{P}\colon \mathsf{P} \hookrightarrow \widehat{\mathsf{P}}$ is the well known *Yoneda embedding* mapping p of P to the presheaf $\mathsf{P}[-, p]$. This extension will be used in Sec. 7.6, in the special case where Q is a presheaf category.

Notation 7.1.2 If $q \leq p$ in a partial order category P, let $[q, p]$ denote the unique arrow in P and $[p, q]$ the unique arrow in P^{op}. We will employ the standard notation [83], writing $x \cdot [q, p]$ for the element $X([p, q])x$, i.e. the restriction of x to the path q.

The categorical presentation of models for concurrency comes with a general notion of *bisimulation from open maps* introduced in [71]. Given a model M, the idea is to identify a *path category* $\mathsf{P} \hookrightarrow \mathsf{M}$ as a subcategory of M. A map $f\colon X \to Y$ in M is then said to be P-*open* (or just open if the path category is clear from the context) if whenever for two path objects P, Q of P and morphism m, p, q such that the diagram



commutes, there exists a morphism $h\colon Q \to X$ as indicated by the dotted line, making the two triangles commute. Intuitively this says that at any point in the simulation described by f, any path extension in Y simulates a corresponding extension in X, i.e. f is like a functional bisimulation. Two objects X and Y is said to be P-bisimilar if they are related by a span of P-open maps f_1, f_2, i.e. $X \xleftarrow{f_1} Z \xrightarrow{f_2} Y$.

From the embedding $\mathcal{Y}_{\mathsf{P}} \colon \mathsf{P} \hookrightarrow \widehat{\mathsf{P}}$, we get a *canonical* path category and thus a canonical notion of bisimulation from open maps for any presheaf category $\widehat{\mathsf{P}}$.

In [71], focus was put on *rooted* presheaves, i.e. presheaves such that $X(\bot)$ is the singleton set if $\bot$ is an initial element of the path category. In particular, it was remarked that the category of rooted presheaves over Act^* is equivalent to the category ST of *synchronisation trees* (with label set Act) and Act^*-bisimulation was shown to coincide with the usual HM-bisimulation [89] on labelled transition systems.

Definition 7.1.3 ([146]) *A* transition system T *(with label set Act) is a quadruple*

$$(S_T, i_T, \longrightarrow_T, Act),$$

where

- S_T *is a set of* states*,*
- $i_T \in S_T$ *is the* initial state*, and*
- $\longrightarrow_T \subseteq S_T \times Act \times S_T$ *is a* transition relation*.*

As usual we write $s \xrightarrow{a}_T s'$ for $\exists (s, a, s') \in \longrightarrow_T$. For a transition $t \in \longrightarrow_T$, let $do(t), co(t), act(t)$ refer to respectively the domain, codomain and action of t. Let $Comp(T) = \{\phi \in \longrightarrow_T^{\infty} | \, \forall 0 < j < |\phi|. co(\phi_{j-1}) = do(\phi_j)\}$, i.e. the set of non-empty (finite or infinite) computations of T and let $Comp_{fin}(T) = Comp(T) \cap \longrightarrow_T^+$, i.e. the finite computations. Define $Run(T) = \{\phi \in Comp(T) \mid do(\phi_0) = i_T\}$, $Run_{fin}(T) = Run(T) \cap \longrightarrow_T^+$ and $Run_{inf}(T) = Run(T) \cap \longrightarrow_T^{\omega}$.

Transition systems (with label set Act) form the objects of a category TS, with arrows being simulations. A simulation from T to T' is a mapping $\sigma \colon S_T \to S_{T'}$ of states, such that

- $\sigma(i_T) = i_{T'}$ *and*
- $s \xrightarrow{a}_T s'$ *implies that* $\sigma(s) \xrightarrow{a}_{T'} \sigma(s')$.

Say a transition system is reachable *if any state is reachable from the initial state.*

A synchronisation tree *is a transition system for which the transition relation is acyclic and any state is reachable from the initial state by a unique sequence of transitions. The synchronisation trees (with label set Act) induces a full subcategory ST of TS.*

The equivalence between rooted presheaves in $\widehat{\mathsf{Act}^*}$ and synchronisation trees is given formally in [147]. Given a rooted presheaf X in $\widehat{\mathsf{Act}^*}$, its corresponding synchronisation tree $\mathcal{E}l(X) = (S_X, i_X, \longrightarrow_X, Act)$ under the equivalence is constructed as follows [1]. The set of states is defined by $S_X = \{(\alpha, x) \mid \alpha \in Act^* \text{ and } x \in X(\alpha)\}$, i.e. the disjoint union of all the sets of elements. The initial state (the root) is given by $i_X = (\bot, *)$, where $\bot$ is the empty sequence in Act^* and $*$ the unique element of $X(\bot)$. There will be a transition $(\alpha, x) \xrightarrow{a} (\alpha a, x')$ iff $x' \cdot [\alpha, \alpha a] = x$, i.e. if $x' \in X(\alpha a)$, $x \in X(\alpha)$, and x' restricts to x.

Note that Act^* is equivalent to the category obtained from Inf by adding a bottom element (the empty sequence). In general, if P is a partial order, let $\mathsf{P}_\bot$ denote the partial order obtained by adding a new bottom element. It is then easy to see, that $\widehat{\mathsf{P}}$ is equivalent to the category of rooted presheaves over $\mathsf{P}_\bot$, so in particular $\widehat{\mathsf{Fin}}$ is equivalent to the category

[1] The category freely generated by the synchronisation tree corresponding to a presheaf X is equivalent to the *category of elements* [83] of X.

ST. Let $\lfloor - \rfloor \colon \widehat{\mathsf{P}} \hookrightarrow \widehat{\mathsf{P}_\perp}$ be the functor mapping a presheaf in $\widehat{\mathsf{P}}$ to its corresponding rooted presheaf in $\widehat{\mathsf{P}_\perp}$. Let $\lceil - \rceil \colon \widehat{\mathsf{P}_\perp} \to \widehat{\mathsf{P}}$ be the converse mapping, discarding the root(s). If $\widehat{\mathsf{P}_\perp}$ is restricted to rooted presheaves this gives the equivalence mentioned above and the open maps between rooted presheaves in $\widehat{\mathsf{P}_\perp}$ via the equivalence are exactly the *surjective* open maps in $\widehat{\mathsf{P}}$. Instead of considering rooted presheaves of a category with bottom, one can thus work with full presheaf categories (not necessarily having a bottom element) and surjective open maps.

By composing the Yoneda embedding with $\lceil - \rceil$ we get an embedding $\mathcal{Y}^\circ_{\mathsf{P}} \colon \mathsf{P}_\perp \hookrightarrow \widehat{\mathsf{P}}$, the *strict extension of* $\mathcal{Y}_{\mathsf{P}}$, mapping the bottom element in $\mathsf{P}_\perp$ to the empty presheaf. In fact $\widehat{\mathsf{P}}, \mathcal{Y}^\circ_{\mathsf{P}}$ is the free connected colimit completion of $\mathsf{P}_\perp$. (A connected colimit is a colimit of a non-empty connected diagram). The following proposition [19, 144, 24] is one of the most important results about open map bisimulation in presheaf models.

Proposition 7.1.4 *Let $F \colon \widehat{\mathsf{P}} \to \widehat{\mathsf{Q}}$ be a connected colimit preserving functor. Then F preserves surjective open maps, i.e. if $m \colon X \to Y$ is surjective open in $\widehat{\mathsf{P}}$ then $F(m) \colon F(X) \to F(Y)$ is surjective open in $\widehat{\mathsf{Q}}$.*

7.1.2 Initial Algebras and Final Coalgebras

Below we recall the categorical analogues of pre- and post-fixed points [10].

Definition 7.1.5 *Let $F \colon \mathsf{P} \to \mathsf{P}$ be an endofunctor on a category P. A* co-algebra *for F is a pair (p, m) of an object and a morphism of P such that $m \colon p \to F(p)$. Dually, an* algebra *for F is a pair (p, m) such that $m \colon F(p) \to p$. The co-algebras of F form the objects of a category $\mathsf{F}_{\mathsf{coAlg}}$, with arrows $f \colon (p, m) \to (q, n)$ being arrows $f : p \to q$ of P such that*

$$\begin{array}{ccc} p & \xrightarrow{m} & F(p) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle F(f)} \\ q & \xrightarrow{n} & F(q) \end{array}$$

commutes. Dually, algebras for F form the objects of a category $\mathsf{F}_{\mathsf{Alg}}$.

Initial and final objects in $\mathsf{F}_{\mathsf{Alg}}$ and $\mathsf{F}_{\mathsf{coAlg}}$ are the categorical analogues of minimal and maximal fixed points of F.

Lemma 7.1.6 *Let $F \colon \mathsf{P} \to \mathsf{P}$ be an endofunctor on a category P. If (p, m) is an* initial algebra *for F, i.e. an initial object in the category of F-algebras, then $m \colon F(p) \to p$ is an isomorphism. Moreover if (q, n) is another initial algebra for F, q is isomorphic to p. The dual statement holds for* final co-algebras*. If F has an initial algebra, let μF denote the (unique up to isomorphism) initial algebra. Similarly, let νF denote the final co-algebra of F if it exists.*

The following lemma is the main technique in proving existence of final co-algebras.

Lemma 7.1.7 *Let P be a category with terminal object $\top$ and $F \colon \mathsf{P} \to \mathsf{P}$ an endofunctor on P. If the ω^{op}-chain*

$$\top \leftarrow F(\top) \leftarrow F^2(\top) \leftarrow \ldots \leftarrow F^n(\top) \leftarrow \ldots$$

has a limiting cone $(P, \{p_n \colon P \to F^n(\top)\}_{n\in\omega})$ *and* F *preserves this limit, i.e.*

$$(F(P), \{! \colon F(P) \to \top\} \cup \{F(p_n) \colon F(P) \to F^{n+1}(\top)\}_{n\in\omega})$$

is a limiting cone too, then the unique mediating (iso)morphism $m \colon P \to F(P)$ *is a final coalgebra.*

The above lemma is the dual of the following lemma for construction of initial algebras, as found in e.g. [10].

Lemma 7.1.8 *Let* P *be a category with initial object* $\bot$ *and* $F \colon \mathsf{P} \to \mathsf{P}$ *an endofunctor on* P*. If the* ω*-chain*

$$\bot \to F(\bot) \to F^2(\bot) \to \ldots \to F^n(\bot) \to \ldots$$

has a colimit P *and* F *preserves this colimit, then the unique mediating (iso)morphism* $m \colon F(P) \to P$ *is an initial algebra.*

Since limits are computed point-wise in a presheaf category $\widehat{\mathsf{P}}$, the terminal object in $\widehat{\mathsf{P}}$ is the presheaf $\top \colon \mathsf{P}^{op} \to \mathsf{Set}$ that yields the one element set (the terminal object in Set) for any object p in P. Dually, the initial object $\bot \colon \mathsf{P}^{op} \to \mathsf{Set}$ is the empty presheaf, yielding the empty set for all objects p in P.

7.2 SCCS, Finite Delay and Fair Parallel

In this section we recall Milner's calculus SCCS [89] of *synchronous* CCS and the definition of a fair parallel composition via a finite delay operator [88]. Assume a distinguished element $1 \in Act$ such that $(Act, \bullet, 1)$ is an Abelian monoid with 1 being the identity. The *basic* operators of SCCS are action prefixing, synchronous product, non-deterministic choice and restriction. Formally, the terms are given by

$$t ::= a{:}t \mid t_1 \times t_2 \mid \Sigma_{i\in I} t_i \mid t\restriction A,$$

where $a \in Act$, $A \subseteq Act$ and I is an index set. With the basic operators we can build processes with only finite behaviour. As usual, we will write 0 for an empty sum, omit the summation sign for a unary sum and write $t_1 + t_2$ for a binary sum.

To be able to define processes with possibly infinite runs, we add a recursion operator, extending the grammar by

$$t ::= \ldots \mid x \mid \mathsf{rec}\,\mathsf{x}.t,$$

where x is a process variable and $\mathsf{rec}\,\mathsf{x}.$ binds the variable x in t. We will let $\mathcal{T}$ refer to the set of closed terms of the calculus SCCS.

The rules given in Fig. 7.1 defines the operational semantics of SCCS, from which we get a *derivation transition system* for any closed term t as defined below.

Definition 7.2.1 *Let* t *be a term in* $\mathcal{T}$*. Then the* derivation transition system *for* t *is the (reachable) transition system* $D(t) = (S, t, \longrightarrow_t, Act)$*, where* $S = \left\{t' \in \mathcal{T} \mid t \longrightarrow^* t'\right\}$*, i.e. all states reachable from* t *by the relation* $\longrightarrow$ *defined by the rules in Fig. 7.1 and* $\longrightarrow_t = \longrightarrow \cap S \times Act \times S$*.*

$$\frac{}{a:t \xrightarrow{a} t}, \quad \frac{t_j \xrightarrow{a} t'}{\Sigma_{i\in I} t_i \xrightarrow{a} t'} \quad (j \in I), \quad \frac{t_1 \xrightarrow{a} t_1' \quad t_2 \xrightarrow{b} t_2'}{t_1 \times t_2 \xrightarrow{a\bullet b} t_1' \times t_2'},$$

$$\frac{t \xrightarrow{a} t'}{t\lfloor A \xrightarrow{a} t'\lfloor A} \quad (a \in A), \quad \frac{t[\mathsf{rec}\,\mathsf{x}.t/x] \xrightarrow{a} t'}{\mathsf{rec}\,\mathsf{x}.t \xrightarrow{a} t'}.$$

Figure 7.1: Operational semantics of SCCS

$$\frac{}{\epsilon t \xrightarrow{1} \epsilon t} \quad \text{(Wait)} \quad \text{and} \quad \frac{t \xrightarrow{a} t'}{\epsilon t \xrightarrow{a} t'} \quad \text{(Fulfill)}.$$

Figure 7.2: Derivation Rules for Finite Delay

Note that in the synchronous product, both processes must perform an action, and the resulting action is the monoid product of the two individual actions. Recursion acts by unfolding and $t[\mathsf{rec}\,\mathsf{x}.t/x]$ is the usual substitution of $\mathsf{rec}\,\mathsf{x}.t$ for the free variable x in t.

An important derived operator introduced in [89] is the *delay* operator δ. For a process t, define $\delta t = \mathsf{rec}\,\mathsf{x}.(1:x+t)$. In the standard semantics, δt is the (unique up to bisimulation) fixed point of the process equation

$$x \sim (1:x+t). \tag{7.3}$$

As an economical way to be able to express that some infinite runs are inadmissible, Milner introduces in [88] a *finite, but unbounded delay* operator ϵ (expectation). Its immediate actions are the same as for the derived delay operator, which can be described by the rules given in Figure 7.2.

However, *infinite* waiting is ruled out as inadmissible. In other words, fulfillment of the delay is always expected. The idea is that finite delay is the *only* operator giving rise to inadmissible infinite runs. Recursion will as usual give rise to admissible infinite runs. This is sufficient to capture *weak fairness* of an *asynchronous* parallel composition. For processes t and t', the fair asynchronous parallel composition [88] of t and t' is defined by $t||t' = (\epsilon t \times t') + (t \times \epsilon t')$. The composition is asynchronous in the sense that one process can delay while the other progress; it is fair in the sense that no process can delay this way forever.

We will let SCCSϵ and $\mathcal{T}_\epsilon$ refer to respectively the calculus SCCS extended with the finite delay operator ϵ and the set of terms of the extended calculus.

In the next section we will introduce two closely related categorical models, suitable for giving respectively denotational and operational semantics in which *inadmissibility* of infinite computations can be expressed.

7.3 Observing Infinite Computations

We approach a categorical model for infinite computations by studying the presheaf model obtained by adding infinite paths to the path category Fin, resulting in the category Inf. This fits with the spirit of [54], where experiments on systems are allowed to consist of *infinite* computations. Categorically, it can be seen as a completion of the path category with all directed colimits.

7.3.1 A Presheaf Model for Infinite Computations

To get a better understanding of presheaves $X\colon \mathsf{Inf}^{op} \to \mathsf{Set}$ in $\widehat{\mathsf{Inf}}$, one can try first to construct a synchronisation tree, as described in Sec. 7.1.1, for the finite part of X, i.e. the restriction of X to Fin. For $\alpha \in Act^{\omega}$, an element $x \in X(\alpha)$ will then specify a unique infinite path in the tree. To be more precise, if $\alpha \in Act^{\omega}$ and $x \in X(\alpha)$ then we will say that x is a *limit point* of the infinite path given by the elements $x \cdot [\beta, \alpha]$ for $\beta \leq_f \alpha$, i.e. the restrictions of x to finite observations. We wish to represent that an infinite path is *admissible* by the *presence* of such a limit point, and that it is *inadmissible* by the *absence* of a limit point. With this interpretation, the model is a bit too general; it allows an infinite path to have two or even more limit points, not representing anything more than if it had only one limit point. We take the subcategory of presheaves with at most one limit point for any infinite sequence as our model. This category is not as ad hoc as it might seem. Actually, it comes about as the category of *separated presheaves* over Inf with respect to a simple Grothendieck topology for Inf, which is often referred to as the sup topology. (In the standard terminology, the infinite paths and limit points are respectively *matching families* and (unique) *amalgations*).

Definition 7.3.1 *Let* $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ *denote the* separated presheaves*, which is the full subcategory of* $\widehat{\mathsf{Inf}}$ *induced by the presheaves* X *satisfying that for all* $x, x' \in X(\alpha)$*,* $\alpha \in Act^{\omega}$

- *(Separated)*$(\forall \beta \leq_f \alpha . x \cdot [\beta, \alpha] = x' \cdot [\beta, \alpha]) \Rightarrow x = x'$.

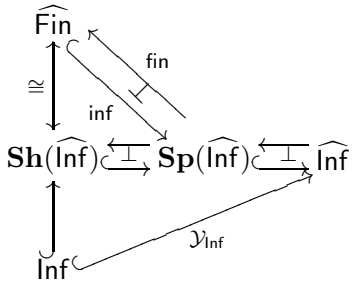
Moreover, we can recover the category $\widehat{\mathsf{Fin}}$ (i.e. of synchronisation trees) within $\widehat{\mathsf{Inf}}$, as being equivalent to the category $\mathbf{Sh}(\widehat{\mathsf{Inf}})$ of *sheaves* over Inf for the same topology. In our case, a separated presheaf is a sheaf if it has *exactly* one limit point for any infinite path. Thus, a sheaf will correspond to a synchronisation tree in which *any* infinite path is admissible, i.e. a *limit closed* synchronisation tree. But this is just the standard interpretation made explicit.

Proposition 7.3.2 *The category* $\widehat{\mathsf{Fin}}$ *is equivalent to the category* $\mathbf{Sh}(\widehat{\mathsf{Inf}})$*, of sheaves over* Inf *with respect to the sup topology.*

Sheaves, separated presheaves and presheaves are known to be closely related and rich in structure [83, 148]. We will especially make use of the fact, that they are related by a sequence of reflections, i.e. the inclusions $\mathbf{Sh}(\widehat{\mathsf{Inf}}) \hookrightarrow \mathbf{Sp}(\widehat{\mathsf{Inf}})$ and $\mathbf{Sp}(\widehat{\mathsf{Inf}}) \hookrightarrow \widehat{\mathsf{Inf}}$ both have left adjoints (reflectors). In our case the reflections are particularly simple. The reflector $\mathsf{sp}\colon \widehat{\mathsf{Inf}} \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$ acts by unifying limit points that specify the same infinite path. The reflector from $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ to $\mathbf{Sh}(\widehat{\mathsf{Inf}})$ acts by completing with limit points of all infinite sequences.

We also have that the objects of Inf under the Yoneda embedding are sheaves. In fact the Grothendieck topology we use is the *canonical* topology for Inf [83], which simply means that it is the largest topology with this property. Together with Prop. 7.3.2, this gives a formal

relationship between the path category Inf, the presheaf model $\widehat{\mathsf{Fin}}$ of finite observations and the models $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ and $\widehat{\mathsf{Inf}}$ of possibly infinite observations as summarized in the diagram below.



(7.4)

Note that this also implies (a general fact) that the category $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ has all limits and colimits. In particular, it shows that limits are computed as in $\widehat{\mathsf{Inf}}$ and similarly for colimits, except for being followed by the reflector, identifying redundant limit points. As indicated in the diagram, we will let $\mathsf{fin} \dashv \mathsf{inf}$ refer to the reflection between $\widehat{\mathsf{Fin}}$ and $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ obtained via the equivalence between $\mathbf{Sh}(\widehat{\mathsf{Inf}})$ and $\widehat{\mathsf{Fin}}$.

For more details on Grothendieck topologies, sheaves and separated presheaves see [83]. The special, and simpler case for a Grothendieck topology on a partially ordered set is given in the appendix, together with the definition of the Grothendieck topology relevant for this paper.

7.3.2 Generalised Transition Systems

A generalised transition systems is a transition system in which the *admissible* infinite computations are represented explicitly. More precisely, we take a generalised transition system to be a transition system together with a set $C \subseteq Comp(T)$ such that $C = C^\bullet$, where $C^\bullet \subseteq Comp(T)$ be the least set including C such that

C1: (composition) if $\phi, \phi' \in C^\bullet$ and $\phi\phi' \in Comp(T)$ then $\phi\phi' \in C^\bullet$,

C2: (pre- and suffix) if $\phi\phi' \in C^\bullet$ and ϕ is finite then $\phi, \phi' \in C^\bullet$ and

C3: (finite) $Comp_{fin}(T) \subseteq C^\bullet$.

The two first conditions ensure that the definition fits with that of *general transition systems* in [54]. The last condition restricts attention to the special case where any finite computation is admissible. It is easy to show that if every state is reachable, the set of admissible computations is determined by a unique set of infinite runs as stated in the lemma below.

Lemma 7.3.3 *Let T be a reachable transition system and $C \subseteq Comp(T)$. If $C = C^\bullet$ then there exists a unique set $A \subseteq Run(T) \backslash Run_{fin}(T)$ such that $C = A^\bullet$.*

Definition 7.3.4 *A* generalised transition system (gts) G (with label set Act) *is a five-tuple $(S_G, Adm_G, i_G, \rightarrow_G, Act)$, such that $T = (S_G, i_G, \rightarrow_G, Act)$ is a transition system (with label set Act) and $Adm_G \subseteq Comp(T)$, the set of* admissible computations*, satisfies that $Adm_G = Adm_G{}^\bullet$. If $G = (S_G, Adm_G, i_G, \rightarrow_G, Act)$ is a generalised transition system let $fin(G) = (S_G, i_G, \rightarrow_G, Act)$, i.e. the underlying transition system. Generalised transition systems (with label set Act) forms the objects of a category GTS. A morphism from G to G' is given by a map $\sigma\colon S_G \rightarrow S_{G'}$ such that*

- $\sigma(i_G) = i_{G'}$,
- $s \xrightarrow{a}_T s'$ *implies that* $\sigma(s) \xrightarrow{a}_{T'} \sigma(s')$ *and*
- $\sigma_\infty(Adm_G) \subseteq Adm_{G'}$,

where σ_∞ *is the map from* $\rightarrow_G^\infty$ *to* $\rightarrow_{G'}^\infty$ *mapping a sequence* $\phi \in \rightarrow_G^\infty$ *to the sequence* ϕ'*, such that* $|\phi| = |\phi'|$ *and for all* $i < |\phi|$*, if* $\phi_i = (s, a, s')$ *then* $\phi'_i = \big(\sigma(s), a, \sigma(s')\big)$*. A* generalised synchronisation tree (gst) *is a generalised transition system for which the underlying transition system is a synchronisation tree. Generalised synchronisation trees (with label set Act) induces a full subcategory* GST *of the category* GTS.

Lemma 7.3.5 *Let* $\sigma\colon S_G \to S_{G'}$ *be a map between the state sets of two generalised transition systems* G *and* G'*. Then the following conditions are equivalent*

1. $\sigma\colon G \to G'$ *is a morphism of generalised transition systems,*
2. - $\sigma(i_G) = i_{G'}$ *and*
   - $\sigma_\infty(Adm_G) \subseteq Adm_{G'}$,
3. - $\sigma\colon fin(G) \to fin(G')$ *is a morphism of transition systems and*
   - $\sigma_\omega(Adm_G \backslash Comp_{fin}(G)) \subseteq Adm_{G'}$,

In particular, the morphisms of GTS restrict to morphisms of the underlying transition systems, so the map fin extends to a functor $fin\colon \mathsf{GTS} \to \mathsf{TS}$. In fact $fin\colon \mathsf{GTS} \to \mathsf{TS}$ is a reflector for the inclusion of TS into GTS that maps a plain transition system to the corresponding *limit closed* generalised transition system (called *standard* in [54]).

Proposition 7.3.6 *The functor* $fin\colon \mathsf{GTS} \to \mathsf{TS}$ *defined on objects in Def.7.3.4 (and leaving morphisms unchanged) is a left adjoint to the inclusion* $inf\colon \mathsf{TS} \hookrightarrow \mathsf{GTS}$ *which maps a transition system* $T = (S_T, i_T, \longrightarrow_T, Act)$ *to the (limit closed) generalised transition system* $(S_T, i_T, \longrightarrow_T, Comp(T), Act)$ *and leaves morphisms unchanged.*

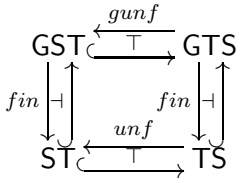
In [146] it is shown that the category ST is a coreflective subcategory of the category TS of transition systems; the inclusion $\mathsf{ST} \hookrightarrow \mathsf{TS}$ is shown to have a right adjoint $unf\colon \mathsf{TS} \to \mathsf{ST}$ which acts on objects by *unfolding* the transition system. This coreflection generalises to one between between GST and a category GTS.

Proposition 7.3.7 *The inclusion functor* $\mathsf{GST} \hookrightarrow \mathsf{GTS}$ *has a right adjoint* $gunf\colon \mathsf{GTS} \to \mathsf{GST}$ *such that the diagram*

$$\begin{array}{ccc} \mathsf{GST} & \xleftarrow{gunf} & \mathsf{GTS} \\ {\scriptstyle fin}\downarrow & & \downarrow{\scriptstyle fin} \\ \mathsf{ST} & \xleftarrow{unf} & \mathsf{TS} \end{array}$$

commutes, where unf *is the unfolding of transition systems defined in [146].*

In fact we have that all four squares in the diagram



commutes.

We will now generalise the equivalence between $\widehat{\mathsf{Fin}}$ and ST mentioned in Sec.7.1 to an equivalence between $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ and GST, giving the promised concrete representation of the presheaves in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$. There is an immediate embedding $e\colon \mathsf{Inf} \hookrightarrow \mathsf{GST}$ of Inf into the category of generalised synchronisation trees (and so the category of generalised transition systems), which maps a finite (or infinite) sequence to the tree with exactly the one corresponding, finite (or infinite, admissible) branch. This gives a *canonical* functor [71] from GTS to $\widehat{\mathsf{Inf}}$, that maps a generalised transition system G to the presheaf $\mathsf{GTS}[e(-), G]$. It is not difficult to check that this will always give a *separated* presheaf.

Lemma 7.3.8 *Let G be a generalised transition system and $e\colon \mathsf{Inf} \hookrightarrow \mathsf{GST} \hookrightarrow \mathsf{GTS}$ the embedding described above. Then $\mathsf{GTS}[e(-), G]$ is a presheaf in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$.*

Restricted to generalised synchronisation trees the canonical functor can equivalently be defined as the functor mapping G to $\mathsf{GST}[e(-), G]$, which gives us one direction of the equivalence.

Theorem 7.3.9 *The categories GST and $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ are equivalent. In one direction the equivalence is given by the (canonical) functor $\mathsf{sps}\colon \mathsf{GST} \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$ that maps a gst G to the separated presheaf $\mathsf{GST}[e(-), G]$. In the other direction the equivalence is given by a functor $\mathcal{E}l\colon \mathbf{Sp}(\widehat{\mathsf{Inf}}) \to \mathsf{GST}$ generalising the functor $\mathcal{E}l\colon \widehat{\mathsf{Fin}} \to \mathsf{ST}$ defined in Sec. 7.1. For X in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$, let $(S, i, \longrightarrow, Act) = \mathcal{E}l(\mathsf{fin}X)$, i.e. the synchronisation tree corresponding to the finite part of X. We then define $\mathcal{E}l(X) = (S, i, \longrightarrow, Adm, Act)$, where*

$$Adm = \{\phi \in \longrightarrow^{\omega} |\ \exists \alpha \in Act^{\omega} \exists x \in \lfloor X \rfloor(\alpha). \forall j \in \omega. do(\phi_j) = \big(\alpha(j), x \cdot [\alpha(j), \alpha]\big)\}^{\bullet}.$$

Note that, restricted to synchronisation trees, the functors fin, inf are just (up to isomorphism) the concrete representation of the reflection between $\widehat{\mathsf{Fin}}$ and $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ given in Diagram (7.4).

7.4 Extended Bisimulation from Open Maps

As described in Sec.7.1, we get a canonical notion of bisimulation from open maps in the presheaf category $\widehat{\mathsf{Inf}}$. From Diagram (7.4) it follows that the notion of Inf-bisimulation restricts to the subcategories $\mathbf{Sh}(\widehat{\mathsf{Inf}})$ and $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ of sheaves and separated presheaves. Since the category Inf can be viewed as a subcategory of the category of generalised transition systems as shown in the previous section, we also get a notion of Inf-bisimulation for generalised transition systems. We show that this bisimulation coincides with the *extended* bisimulation

defined for general transition systems in [54]. Since Inf-bisimulation for generalised synchronisation trees coincides with the Inf-bisimulation in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ this gives a concrete representation of the canonical bisimulation in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ as well.

First let us give a characterisation of the Inf-open maps of GTS, generalising the "zig-zag" morphisms in [71].

Proposition 7.4.1 *Let $T = (S, i, \longrightarrow, Adm, Act)$ and $U = (S_U, i_U, \longrightarrow_U, Adm_U, Act)$ be generalised transition systems and $\sigma\colon T \to U$. Then σ is* Fin*-open if and only if for all reachable states s of T*

- *if $\sigma(s) \xrightarrow{a}_U s'_1$ then $s \xrightarrow{a} s_1$ and $\sigma(s_1) = s'_1$ for some state $s_1 \in S$,*

and σ is Inf*-open if and only if moreover*

- *if $\phi' \in Adm_U$ and $\phi' = \sigma(s) \xrightarrow{a_1}_U s'_1 \xrightarrow{a_2}_U s'_2 \xrightarrow{a_3}_U \ldots \xrightarrow{a_n}_U s'_n \xrightarrow{a_{n+1}}_U \ldots$ then there exists $\phi \in Adm$ such that $\phi = s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} s_n \xrightarrow{a_{n+1}} \ldots$ and for all $j \in \omega$, $\sigma(s_j) = s'_j$*

Now we give the definition of extended bisimulation from [54] reformulated as a relation between two generalised transition systems (and exploiting condition C3).

Definition 7.4.2 *([54]) Let T and T' be generalised transition systems. Then T and T' are* extended bisimilar *if there exists a relation $R \subseteq S_T \times S_{T'}$ such that $(i_T, i_{T'}) \in R$ and if $(s, s') \in R$ then*

E1. *if there exists a computation $\phi \in Adm_T$ s.t. $\phi_0 = s$, then there exists a computation $\phi' \in Adm_{T'}$ s.t. $|\phi| = |\phi'|$ and $\phi'_0 = s'$ and for $0 \leq j < |\phi|$, $act(\phi_j) = act(\phi'_j)$ and $(\phi_j, \phi'_j) \in R$,*

E2. *if there exists a computation $\phi' \in Adm_{T'}$ s.t. $\phi'_0 = s'$, then there exists a computation $\phi \in Adm_T$ s.t. $|\phi| = |\phi'|$ and $\phi_0 = s$ and for $0 \leq j < |\phi|$, $act(\phi_j) = act(\phi'_j)$ and $(\phi_j, \phi'_j) \in R$,*

Note that (by condition C3) extended bisimulation specialises to the standard HM-bisimulation on transition systems if only sequences ϕ and ϕ' of length one is considered in E1 and E2. Also note that (by the conditions C1 and C2) one could equivalently have formulated the bisimulation considering only sequences being infinite or of length one. From these considerations and Prop. 7.4.1 it follows that extended bisimulation coincides with Inf-bisimulation for generalised transition systems.

Proposition 7.4.3 *Let G and G' be generalised transition systems. Then G and G' are* Inf*-bisimilar if and only if G and G' are extended bisimilar.*

It is an easy fact that Inf-bisimulation in GST under the equivalence coincides with Inf-bisimulation in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$, so we get the following corollary.

Corollary 7.4.4 *Let X and X' be presheaves in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$. Then X and X' are* Inf*-bisimilar if and only if $\mathcal{E}l(X)$ and $\mathcal{E}l(X')$ are extended bisimilar.*

Remark that from the coreflection given in the previous section and Lem. 6 in [71] it follows that two generalised transition systems are Inf-bisimilar if and only if their unfoldings as generalised synchronisation trees are Inf-bisimilar.

$$\frac{}{\epsilon_n t \xrightarrow{1} \epsilon_{n+1} t} \quad \text{(Wait)} \quad \text{and} \quad \frac{t \xrightarrow{a} t'}{\epsilon_n t \xrightarrow{a} t'} \quad \text{(Fulfill)}.$$

Figure 7.3: Derivation rules for annotated finite delay

7.5 Operational Semantics

In this section we will express Milner's operational semantics of SCCS with finite delay [88] in terms of generalised transition system. First the two rules in Figure7.2 are added to the rules of Figure7.1. Next the inadmissible infinite computations are identified via the notions of waiting computations, subagents and subcomputations. Put briefly: A computation $t_0 \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \ldots$ of an agent t_0 is waiting if $t_i = \epsilon t$ for all i and every transition is inferred *solely* from the (Wait) rule for finite delay. Agents $a:t$, $\mathsf{rec}\,x.t$, $\Sigma_{i\in I} t_i$ and ϵt have only themselves as subagent, $t\lfloor A$ has the subagents of t and $t_1 \times t_2$ has the subagents of t_1 and t_2. Any computation of an agent t is then inferred from computations of the subagents, which are referred to as subcomputations. A computation is defined to be admissible if it is finite or has no sequel (i.e. suffix) with an infinite waiting subcomputation.

To define a derivation transition system in which we can distinguish admissible from inadmissible infinite runs we thus need to record if the (Wait) rule was used to infer an action of a subagent. Consequently, we will annotate terms of the form ϵt with a number $n \in \omega$ written $\epsilon_n t$, which indicates for how long they have been delaying. In the following $\mathcal{T}_\epsilon$ will generally refer to the set of annotated closed terms of SCCSϵ. Note that any function with domain $\mathcal{T}$ can be regarded as a function with domain $\mathcal{T}_\epsilon$ by discarding the annotations. For simplicity we will let $\epsilon_0 t$ and ϵt refer to the same agent. The derivation rules of Figure7.2 is then replaced by the rules in Figure7.3.

The *position* of a subagent is formalised as follows.

Definition 7.5.1 *Define $Pos = \{1,2\}^*$, a set of* positions*, and let $nil \in Pos$ denote the empty sequence (the top position). Any term t in $\mathcal{T}_\epsilon$ define a partial function $t\colon Pos \rightharpoonup \mathcal{T}_\epsilon$, given inductively (in the length of the position and the structure of t) by*

$$t(nil) = \begin{cases} t & \text{if } t \equiv a{:}t',\ t \equiv \mathsf{rec}\,x.t',\ t \equiv \Sigma_{i\in I} t_i \text{ or } t \equiv \epsilon t' \text{ for some } t', \\ t'(nil) & \text{if } t \equiv t'\lfloor A, \\ undef & \text{otherwise}, \end{cases}$$

$$t(ip) = \begin{cases} t_i(p) & \text{if } t \equiv t_1 \times t_2, \\ t'(ip) & \text{if } t \equiv t'\lfloor A, \\ undef & \text{otherwise.} \end{cases}$$

For $p \in Pos$ and t an annotated term, we will say that $t(p)$ is waiting *if $t(p) = \epsilon_n t'$ for some term t' and $n > 1$.*

Now, we can define when an infinite computation is inadmissible.

Definition 7.5.2 *An infinite computation $t_0 \xrightarrow{\alpha_0} t_1 \xrightarrow{\alpha_1} t_2 \xrightarrow{\alpha_2} \ldots$ derivable by the rules in Figure7.1 and Figure7.3 is* inadmissible *if and only if there exist $j \in \omega$ and a position $p \in$*

$\{1,2\}^*$ *such that* $\forall j' \geq j$, $t_{j'}(p)$ *is waiting. We say that a computation is* admissible *if it is not inadmissible.*

It is not difficult to verify that a computation is inadmissible by the definition above if and only if it has a suffix with a waiting subagent which continues to wait forever, so the definition of admissibility coincides with that of [88] which we briefly gave in the beginning of the section.

The derivation transition systems for terms in $\mathcal{T}_\epsilon$ are generalised transition systems with the set of admissible computations given by Def. 7.5.2 above.

Definition 7.5.3 *Let t be a term in $\mathcal{T}_\epsilon$. Then the* derivation transition system *for t is the reachable* generalised *transition system $\mathcal{O}_\epsilon(t) = (S, t, \longrightarrow_t, Adm, Act)$, where $S = \{t' \mid t \rightarrow^* t'\}$, $\longrightarrow_t = \rightarrow \cap\, S \times Act \times S$ is the relation defined by the rules in Figure7.1 and Figure7.3 restricted to states in S, and $Adm \subseteq Comp\big((S, t, \longrightarrow_t, Act)\big)$ is the set of admissible computations as defined in Def. 7.5.2.*

Remark 7.5.4 Though it is not important for the present paper, note that we do not need to record *exactly* how many steps a delay has waited, just if has waited zero, one or more than one step continuously. This means that we could replace the first rule in Figure7.3 by the rule $\epsilon_n t \xrightarrow{1} \epsilon_{\min\{n+1,2\}} t$ and only allow the numbers 0, 1 and 2 in annotations. The latter set of rules has the benefit of not giving rise to infinite graphs just because of the presence of a finite delay, which e.g. could be relevant in connection with model checking.

7.6 Presheaf Semantics

In this section we will see that the category of separated presheaves $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ is well suited to give denotational semantics to SCCSϵ.

7.6.1 Semantics of Basic Operators

The denotation of sum is simply given by the coproduct in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$. The denotations of the remaining basic operators, restriction, action prefix, and synchronous product, can be obtained from the underlying functions on sequences using the free extension $(-)_!$ described in Sec. 7.1, in the case where $\mathsf{Q} = \mathbf{Sp}(\widehat{\mathsf{Inf}})$.

For $A \subseteq Act$, the *restriction on sequences* $(-)\restriction A \colon \mathsf{Inf} \to \mathsf{Inf}_\perp$ maps a sequence α to the (possible empty) sequence $\alpha' \leq \alpha$ being the longest prefix of α in A^*, i.e. the sequence $\alpha' \leq \alpha$ such that if $\alpha = \alpha' a \alpha''$ then $a \notin A$.

For $a \in Act$, the *action prefix on sequences* $a \colon \mathsf{Inf}_\perp \to \mathsf{Inf}$ maps a (possibly empty) sequence α to $a\alpha$.

The *synchronous product on sequences*, $\bullet \colon \mathsf{Inf} \times \mathsf{Inf} \to \mathsf{Inf}$ is the extension of the monoid product to sequences, i.e. for $\alpha, \beta \in \mathsf{Inf}$, $\alpha \bullet \beta = \gamma$, where γ is the unique sequence such that $|\gamma| = \min\{|\alpha|, |\beta|\}$ and $\gamma_i = \alpha_i \bullet \beta_i$.

It is easy to see that the above mappings are monotone, and thus functors between partial order categories. By (implicitly) composing with the embeddings $\mathcal{Y}^\circ_{\mathsf{Inf}_\perp} \colon \mathsf{Inf}_\perp \hookrightarrow \mathbf{Sp}(\widehat{\mathsf{Inf}})$ and $\mathcal{Y}_{\mathsf{Inf}} \colon \mathsf{Inf} \hookrightarrow \mathbf{Sp}(\widehat{\mathsf{Inf}})$, we get functors $(-)\restriction A \colon \mathsf{Inf} \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$, $a \colon \mathsf{Inf}_\perp \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$ and $\bullet \colon \mathsf{Inf} \times \mathsf{Inf} \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$. Applying the extension $(-)_!$ we get the following denotations of basic operators.

Basic operators: For closed terms t, t' and t_i, define

$$\mathcal{I}[\![\Sigma_{i\in I} t_i]\!] = \Sigma_{i\in I}\mathcal{I}[\![t_i]\!], \tag{7.5}$$
$$\mathcal{I}[\![a\!:\!t]\!] = \mathsf{sp}(a_! \circ \lfloor \mathcal{I}[\![t]\!] \rfloor), \tag{7.6}$$
$$\mathcal{I}[\![t \times t']\!] = \mathsf{sp}(\mathcal{I}[\![t]\!](\bullet_! \circ w)\mathcal{I}[\![t']\!]), \tag{7.7}$$
$$\mathcal{I}[\![t \lfloor A]\!] = \mathcal{I}[\![t]\!] \lfloor A_!, \tag{7.8}$$

where $a_!\colon \widehat{\mathsf{Inf}_\perp} \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$ is precomposed with the lifting functor $\lfloor - \rfloor\colon \widehat{\mathsf{Inf}} \hookrightarrow \widehat{\mathsf{Inf}_\perp}$ defined in Sec. 7.1 and $\bullet_!\colon \widehat{\mathsf{Inf} \times \mathsf{Inf}} \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$ is precomposed with the (connected colimit-preserving [22]) functor $w\colon \widehat{\mathsf{Inf}} \times \widehat{\mathsf{Inf}} \to \widehat{\mathsf{Inf} \times \mathsf{Inf}}$ defined (on objects) by $w(X,Y)(\alpha,\beta) = X(\alpha) \times Y(\beta)$. The semantic functions are extended in the obvious way to terms t with free variables in a set $\mathcal{V}$, yielding functors

$$\mathcal{I}[\![t]\!]_{\mathcal{V}}\colon \prod_{x\in\mathcal{V}} \mathbf{Sp}(\widehat{\mathsf{Inf}}) \to \mathbf{Sp}(\widehat{\mathsf{Inf}}).$$

Since the functors are build up from connected colimit preserving functors it follows that they themselves preserve connected colimits.

The first three definitions (7.5)-(7.7) above only give the denotation up to isomorphism. It is helpful, e.g. in showing correspondence with the operational semantics, to give an explicit semantics $[\![t]\!]$ such that $[\![t]\!] \cong \mathcal{I}[\![t]\!]$. We will just give the action on objects. The tags sum and $\times$ are used to indicate clearly how an element came about, which we will use in App. 7.9.

$$[\![\Sigma_{i\in I} t_i]\!]\alpha = \big\{(\mathsf{sum}\ i, (\alpha, e)) \mid i \in I \text{ and } e \in [\![t_i]\!]\alpha\big\}. \tag{7.9}$$

$$[\![a\!:\!t]\!]\alpha = \begin{cases} \lfloor [\![t]\!]\alpha' & \text{if } \alpha = a\alpha', \\ \varnothing & \text{otherwise,} \end{cases} \tag{7.10}$$

where we choose to represent $\lfloor - \rfloor\colon \widehat{\mathsf{Inf}} \hookrightarrow \widehat{\mathsf{Inf}_\perp}$ explicitly by

$$\lfloor X \rfloor\alpha = \begin{cases} \{*\} & \text{if } \alpha = \perp, \\ X\alpha & \text{otherwise.} \end{cases} \tag{7.11}$$

$$\begin{aligned} [\![t_1 \times t_2]\!]\alpha = \{&(\beta, e_1) \times (\gamma, e_2) \mid \\ &\beta, \gamma \in \mathsf{Inf}.\beta \bullet \gamma = \alpha \text{ and } e_1 \in [\![t_1]\!]\beta \text{ and } e_2 \in [\![t_2]\!]\gamma\}. \end{aligned} \tag{7.12}$$

$$[\![t \lfloor A]\!]\alpha = \{e \mid \alpha \in A^\infty \text{ and } e \in [\![t]\!]\alpha\}. \tag{7.13}$$

7.6.2 Semantics of Recursion

For *recursion* we need to take care. In a "standard" semantics one would take least fixed points, i.e. initial algebras as the meanings of recursion. However in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$, this would not reflect that it is admissible to unfold a recursion infinitely. An explicit example that illustrates this is given below, showing that the initial algebra of the functor corresponding

to the delay equation given in Sec. 7.2 will be the proper denotation of *finite* delay and not the delay operator derived using recursion. The solution is to take *final* co-algebras as the meanings of recursion.

Infinite recursion: For a term t with one free variable x, define

$$\mathcal{I}[\![\mathsf{rec}\,\mathsf{x}.t]\!] = \nu\mathcal{I}[\![t]\!],$$

i.e. (the object of) a final co-algebra of the endofunctor $\mathcal{I}[\![t]\!]\colon \mathbf{Sp}(\widehat{\mathsf{Inf}}) \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$. For this to be well defined, we must show existence of final co-algebras for all functors. We will use Lem. 7.1.7 given in Sec. 7.1 to construct final co-algebras for all relevant endofunctors as limits of ω^{op}-chains. The definition is then extended to processes with more than one variable in the usual way as a limit with parameters [82]. From the explicit definitions given in Eq. (7.13)-(7.12) we can show that all basic operators preserve ω^{op}-limits. From the general fact that limits commute with limits [82] we get that recursion preserves ω^{op}-limits as well, i.e. if $\mathsf{rec}\,\mathsf{x}.t$ has free variables then $\mathcal{I}[\![\mathsf{rec}\,\mathsf{x}.t]\!]$ preserves ω^{op}-limits.

Lemma 7.6.1 *Let t be a (possibly open) term of SCCS with free variables in $\mathcal{V}$. If*

$$\mathcal{I}[\![t]\!]_{\mathcal{V}}\colon \prod_{x\in\mathcal{V}} \mathbf{Sp}(\widehat{\mathsf{Inf}}) \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$$

(is well defined and) preserves ω^{op}-limits then

$$\mathcal{I}[\![t\lfloor A]\!]_{\mathcal{V}}\colon \prod_{x\in\mathcal{V}} \mathbf{Sp}(\widehat{\mathsf{Inf}}) \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$$

(is well defined and) preserves ω^{op}-limits, and similarly for sum, prefix, synchronous product and recursion.

As for the basic operators, we can give an explicit denotation of recursion $[\![\mathsf{rec}\,\mathsf{x}.t]\!] \cong \mathcal{I}[\![\mathsf{rec}\,\mathsf{x}.t]\!]$. First we choose an explicit representation of a final presheaf $\top$ by defining $\top\alpha = \{*\}$. Now we use the explicit definition of limits in the category Set to define

$$[\![\mathsf{rec}\,\mathsf{x}.t]\!]\alpha = \big\{\langle e_0, e_1, \ldots, e_n, \ldots\rangle \in \prod_{n\in\omega} [\![t]\!]^n(\top)\alpha \mid [\![t]\!]^n(\tau)_\alpha e_{n+1} = e_n\big\}, \tag{7.14}$$

where $\tau\colon [\![t]\!](\top) \to \top$ is the natural transformation given by $\tau_\alpha(e) = *$ for any $e \in [\![t]\!](\top)\alpha$. We have projections $\pi_n\colon [\![\mathsf{rec}\,\mathsf{x}.t]\!] \to [\![t]\!]^n(\top)$ and by universality we get an (explicit) isomorphism $\rho_t\colon [\![\mathsf{rec}\,\mathsf{x}.t]\!] \to [\![t]\!]([\![\mathsf{rec}\,\mathsf{x}.t]\!])$, such that

$$\begin{array}{ccc} [\![\mathsf{rec}\,\mathsf{x}.t]\!] & \xrightarrow{\pi_{n+1}} & [\![t]\!]^{n+1}(\top) \\ {\scriptstyle\rho_t}\downarrow & \nearrow_{[\![t]\!](\pi_n)} & \\ [\![t]\!]([\![\mathsf{rec}\,\mathsf{x}.t]\!]) & & \end{array} \tag{7.15}$$

commutes for any $n \in \omega$. Note that, in general if t has free variables $\mathcal{V} \uplus \{x\}$ then ρ_t and π_n are natural transformations.

We have now given semantics to all operators in SCCSϵ except for finite delay. It is worth remarking, that already at this stage it is clear that this semantics will not (in general) correspond to the operational semantics given in Sec. 7.5. A simple example showing this is

provided by the (disastrous) term $\mathsf{rec}\,\mathsf{x}.x$. According to the operational semantics, this term denotes the process that cannot perform any actions, which is also the process denoted by the empty sum 0. It is not difficult to compute the appropriate limit finding that $\mathcal{I}[\![\mathsf{rec}\,\mathsf{x}.x]\!] \cong \top$, i.e. (the) final object in $\widehat{\mathsf{Inf}}$, which in no sensible way can be equated to the denotation of the empty sum, which is the *initial* object in $\widehat{\mathsf{Inf}}$. (Note that this is indeed the result if one constructs the initial algebra instead).

However, as we will see below, we get the desired correspondence if we restrict the language to only allow *guarded* recursion.

7.6.3 Semantics of Finite Delay

As mentioned above, the denotation of finite delay comes about as the *initial* algebra of the functor corresponding to the delay equation.

Finite delay: For a closed term t, define

$$\mathcal{I}[\![\epsilon t]\!] = \mu\mathcal{I}[\![1 : x + t]\!],$$

i.e. (the object of) an initial algebra of the endofunctor $\mathcal{I}[\![1 : x + t]\!] \colon \mathbf{Sp}(\widehat{\mathsf{Inf}}) \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$. This initial algebra exists by Lem. 7.1.8 since the denotation of prefixing preserves connected colimits and the denotation of sum all colimits. The definition is extended to open terms (in which t is not free) as a colimit with parameters.

From the explicit definition of colimits in Set, we find that we can take

$$[\![\epsilon t]\!]\alpha = \left\{ \big(\mathsf{del}\ n, (\alpha', e)\big) \mid n \in \omega,\ \alpha = 1^n\alpha' \text{ and } e \in \lfloor [\![t]\!] \rfloor \alpha' \right\} \tag{7.16}$$

as explicit definition of finite delay on objects (again the tag del is used to indicate clearly that the element arise from the denotation of a finite delay). For $\beta \leq \alpha$, define $[\![\epsilon t]\!]([\alpha, \beta])$ by

$$\big(\mathsf{del}\ n, (\alpha', e)\big) \cdot [\beta, \alpha] = \begin{cases} \big(\mathsf{del}\ n, (\beta', e \cdot [\beta', \alpha'])\big) & \text{if } \beta = 1^n\beta', \\ \big(\mathsf{del}\ m, (\bot, *)\big) & \text{if } \beta = 1^m \text{ for } m < n, \end{cases}$$

for $n \in \omega$, $\alpha = 1^n\alpha'$ and $e \in \lfloor [\![t]\!] \rfloor \alpha'$.

To guarantee that the denotation of recursion is still well-defined, we need to check that the denotations of finite delay preserve ω^{op}-limits. This can be done from the explicit definition given above.

Lemma 7.6.2 *Let t be a (possibly open) term of SCCSϵ with free variables in $\mathcal{V}$. If*

$$\mathcal{I}[\![t]\!]_{\mathcal{V}} \colon \prod_{x \in \mathcal{V}} \mathbf{Sp}(\widehat{\mathsf{Inf}}) \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$$

(is well defined and) preserves ω^{op}-limits then

$$\mathcal{I}[\![\epsilon t]\!]_{\mathcal{V}} \colon \prod_{x \in \mathcal{V}} \mathbf{Sp}(\widehat{\mathsf{Inf}}) \to \mathbf{Sp}(\widehat{\mathsf{Inf}})$$

(is well defined and) preserves ω^{op}-limits.

This completes the definition of our denotational semantics of SCCSϵ in the category of separated presheaves $\mathbf{Sp}(\widehat{\mathsf{Inf}})$.

7.6.4 Extended Bisimulation Congruence

From the fact that the denotations (in $\widehat{\mathsf{Inf}}$) of all basic operators are built from connected colimit preserving functors, it follows that they preserve open maps in $\widehat{\mathsf{Inf}}$. Using the fact that the inclusion of $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ in $\widehat{\mathsf{Inf}}$ is full, together with proposition 5 in [71] we get that this holds in $\mathbf{Sp}(\widehat{\mathsf{Inf}})$ as well. It is easy to show from the explicit definition that the denotations of finite delay preserve open maps as well (alternatively one could use the same technique as used in [22] for showing that denotations of recursions (given by initial algebras) preserve open maps).

Proposition 7.6.3 *Extended bisimulation is a congruence with respect to all basic operators of SCCSϵ as well as finite delay.*

However, when it comes to *recursion* we meet a problem: What is the "right" notion of bisimulation (from open maps) for denotations of open terms, i.e. functors between presheaf categories? In [22] the notion of open maps is extended to open natural transformations, being natural transformations for which all components are open maps. This is shown to be sufficient to guarantee that open map bisimulation is a congruence with respect to the denotations of recursion (given by *initial* algebras) in a CCS-like calculus. In [144, 24] is suggested a slightly stronger notion of open maps between (connected) colimit preserving functors between presheaf categories which them self can be regarded as objects of a presheaf category and thus comes with a canonical notion of open maps. The second notion requires all functors to be (connected) colimit preserving functors, which is not known to be the case in our setting (because of the use of final co-algebras). The notion of open natural transformations could be used, but we have not yet been able to show that it is sufficient to give the desired congruence property.

7.6.5 Full Abstraction

Using the representation theorem in Sec. 7.3 we can express the denotational semantics given above in terms of generalised synchronisation trees, defining $\mathcal{D}_\epsilon(t) = \mathcal{E}l([\![t]\!])$. This allows us to relate the denotational semantics directly to the operational semantics given in Sec. 7.5 within the category GTS. First of all we will restrict attention to terms with only *guarded recursion*. Recall from e.g. [89] that a recursion $\mathsf{rec}\,x.t$ is guarded, if all free occurrences of x in t is guarded, that is, within a subterm $a:t'$ of t for some action $a \in Act$. Let $\mathcal{T}_g$ refer to the set of all closed, possibly annotated terms of SCCSϵ with only guarded recursion. We will say that a term t in $\mathcal{T}_g$ is *standard* if for all subterms $e_n t'$ it holds that $n = 0$. We will then show, that if we quotient by open map bisimulation, the denotational semantics for standard terms in $\mathcal{T}_g$ is in fact *equationally fully abstract* with respect to extended bisimulation. This means that for any two standard terms t and t' of $\mathcal{T}_g$, the presheaves $[\![t]\!]$ and $[\![t']\!]$ are bisimilar if and only if the generalised transition systems $\mathcal{O}_\epsilon(t)$ and $\mathcal{O}_\epsilon(t')$ arising from the operational semantics are extended bisimilar. As remarked in Sec. 7.6.2 above, we cannot obtain this result for all terms of SCCSϵ.

The proof (see App. 7.9 for a more detailed proof outline) goes by showing that there exists an Inf-open morphism of generalised transition systems from $\mathcal{D}_\epsilon(t)$ to $\mathcal{O}_\epsilon(t)$ for any term t in $\mathcal{T}_g$.

Proposition 7.6.4 *Let t be a standard term in $\mathcal{T}_g$. Then there exists an* Inf*-open morphism of generalised transition systems $F_t \colon \mathcal{D}_\epsilon(t) \to \mathcal{O}_\epsilon(t)$.*

From the proposition above and Prop. 7.4.3 and Cor. 7.4.4 in Sec. 7.4 we can now deduce the desired result.

Theorem 7.6.5 *Let t and t' be terms in $\mathcal{T}_g$. Then $[\![t]\!]$ and $[\![t']\!]$ are open map bisimilar if and only if $\mathcal{O}_\epsilon(t)$ and $\mathcal{O}_\epsilon(t')$ are extended bisimilar.*

7.7 Conclusion and Future Work

This paper has two main contributions. The first is a generalisation of the categorical models for concurrency as developed in [146, 71, 24], providing both a generalised transition system and a presheaf model for *infinite* computations, suitable for agents with a notion of *fairness* or *inadmissible* infinite computations. The generalised transition systems are instances of those proposed in [54] and the *extended bisimulation* given there is shown to coincide with the abstract bisimulation from span of open maps in our model. The second main contribution is that we give both an operational semantics and a denotational semantics for SCCS with finite delay, representing the notion of inadmissible infinite *computations* precisely as given in [88] allowing behaviours to be discriminated up to *extended bisimulation*. This notion of bisimulation is a strictly finer, and as argued in the present paper and in [7], more intuitive, equivalence than the one obtained from the fortification preorder in [88], which except for [7] has been the basis for previous semantics of SCCS with finite delay [53, 65, 64]. Benefitting from the categorical presentation, our semantics appears to give a conceptually simpler treatment of infinite computations than the one in [7].

A number of questions remains to be explored. An obvious question is if one could generalise the finite delay to a *fair recursion* as in [64]. Work is in progress on a notion of open maps between denotations of open terms stronger than the one in [22], for which open map bisimulation is a congruence with respect to recursion. We get a characteristic HML-like path logic [71] for extended bisimulation from the open maps approach, which should be compared to the characteristic logic given in [54]. Here comes the question about decidability of extended bisimulation. If one restricts attention to agents for which products and restrictions are disallowed within recursions and change the operational semantics according to the remark in Sec. 7.5 all agents will be assigned *finite* (generalised) transition systems. It would be interesting to explore if there is any relationship between the present approach and the more traditional domain theoretical approach to fairness and countable non-determinism as in e.g. [108]. Finally, we hope to be able to extend the presheaf model for (finitary) dataflow given in [59] to infinite computations along the lines of the present paper, giving a model of dataflow in which fairness, maybe even *fair merge* [101], can be expressed.

Acknowledgements: Thanks to Glynn Winskel, Marcelo Fiore and Prakash Panangaden for helpful and encouraging discussions.

7.8 Grothendieck topology for a partial order

Here we give the definitions from [83] of a *Grothendieck topology* for a category P and the *sup* topology, specialised to the case where P is a partial order. Let P be a partial order and

$p \in P$. Define $p{\downarrow} = \{p' \in P \mid p' \leq p\}$. A *sieve* S on p is then a set $S \subseteq p{\downarrow}$, i.e. a downwards closed set below p.

Definition 7.8.1 (Grothendieck topology for a partial order) *A* Grothendieck topology for a partial order P*, is a function J which assigns to each object p of P a set $J(p)$ of sieves on p, in such a way that*

C1*: (maximal sieve) $p{\downarrow} \in J(p)$,*

C2*: (stability) if $S \in J(p)$ and $q \leq p$ then $q{\downarrow} \cap S \in J(q)$,*

C3*: (transitivity) if $S \in J(p)$ and R is any sieve on p such that $q{\downarrow} \cap R \in J(q)$ for all $q \in S$, then $R \in J(p)$.*

Assume J is a topology for a partial order P. We will now describe when a presheaf $X \colon \mathsf{P}^{\mathsf{op}} \to \mathsf{Set}$ in $\widehat{\mathsf{P}}$ is a sheaf with respect to J. Assume p is an element of P and $S \in J(p)$, i.e. a *sieve covering* p. A *matching family* for S of elements of X is a function that assigns to each element $q \in S$ an element $x_q \in X(q)$ such that $x_q \cdot [r, q] = x_r$ for any $r \leq q$. Given such a matching family, an element $x \in X(p)$ is an *amalgation*, if $x \cdot [q, p] = x_q$ for all $q \in S$. Then X is respectively a *separated presheaf* or a *sheaf* with respect to J if for any object $p \in P$, any matching family for any sieve $S \in J(p)$ has respectively *at most one* or *a unique* amalgation.

Definition 7.8.2 (separated presheaves and sheaves) *For a partial order P and a Grothendieck topology J on P, let $\mathbf{Sp}_J(\widehat{\mathsf{P}})$ and $\mathbf{Sh}_J(\widehat{\mathsf{P}})$ be the full subcategories of $\widehat{\mathsf{P}}$ induced by respectively the* separated presheaves *and the* sheaves *with respect to J. If the topology J is clear from the context, we will just write respectively $\mathbf{Sp}(\widehat{\mathsf{P}})$ and $\mathbf{Sh}(\widehat{\mathsf{P}})$.*

For a sequence α in Inf (as defined in Sec. 7.1), a sieve on α is simply a prefix closed set of sequences below α. We only use the *sup* topology on Inf, which to each sequence α assigns the set $\{S \mid S \text{ is a sieve on } \alpha \text{ and } \bigsqcup S = \alpha\}$, i.e. of all sieves that have α as supremum. It is easy to check that this satisfy the conditions in Def. 7.8.1, and that it works for any partial order. This topology is in fact the *canonical* topology for Inf, being the largest topology such that $\mathcal{Y}_{\mathsf{Inf}}\alpha$ is a sheaf for any α.

Definition 7.8.3 (sup topology for Inf) *For the partial order Inf, the* sup topology *J is given by $J(\alpha) = \{\alpha{\downarrow}, \{\beta \mid \beta \leq_f \alpha\}\}$, for $\alpha \in \mathsf{Inf}$*

Note that if α is finite then $J(\alpha)$ contains just $\alpha{\downarrow}$, i.e. the maximal sieve on α.

7.9 Proof of Full Abstraction

We will here give a more detailed proof outline for Prop. 7.6.4 of Sec. 7.6.5 as repeated below. Recall that $\mathcal{T}_g$ refer to the set of all closed terms of SCCSϵ with only guarded recursion and that a term is standard if for all subterms $\epsilon_n u''$, $n = 0$. Let $\mathcal{T}_g^o$ refer to the set of, possible open, terms of SCCSϵ with only guarded recursion.

Proposition 7.9.1 (Prop. 7.6.4 of Sec. 7.6.5) *Let t be a standard term in $\mathcal{T}_g$. Then there exists an Inf-open morphism of generalised transition systems $F_t \colon \mathcal{D}_\epsilon(t) \to \mathcal{O}_\epsilon(t)$.*

We will need some preliminary definitions. For t a term in SCCSϵ, $FV(t)$ will denote the set of free variables in t. As in [88] [2] we define $gd(t)$, the *guard-depth* of t by

- $gd(x) = gd(a{:}t) = 0$,
- $gd(\Sigma_{i\in I} t_i) = \sup\{gd(t_i) + 1 \mid i \in I\}$,
- $gd(t_1 \times t_2) = \max\{gd(t_1) + 1, gd(t_2) + 1\}$, and
- $gd(\mathsf{rec}\,\mathsf{x}.t) = gd(t\lfloor A) = gd(\epsilon t) = gd(t) + 1$.

This is a well defined ordinal, but not necessarily a finite number because sums can be infinite. As in [88] the following is a key property of gd for use in inductive proofs in the guard depth of terms with only guarded induction.

Lemma 7.9.2 *If x is guarded in t then $gd(t[t'/x]) = gd(t)$.*

Proof. By a straightforward structural induction. □

For a term t in $\mathcal{T}_\epsilon$ we define $sd(t)$, the *subagent depth* of t by

- $sd(a{:}t) = sd(\Sigma_{i\in I} t_i) = sd(\mathsf{rec}\,\mathsf{x}.t) = sd(\epsilon t) = 0$,
- $sd(t_1 \times t_2) = 1 + \max\{sd(t_1), sd(t_2)\}$, and
- $sd(t\lfloor A) = 1 + sd(t)$.

This is simply the maximal depth of a subagent and thus always finite.

For a gst $T = (S, i, \longrightarrow, Adm, Act)$ and $s \in S$ we define *the gst above s in T* by $T_{s\lhd} = (S_{s\lhd}, s, \longrightarrow_{s\lhd}, Adm_{s\lhd}, Act)$, where

- $S_{s\lhd} = \{s' \mid s \longrightarrow^* s'\}$,
- $\longrightarrow_{s\lhd} = \longrightarrow \cap\, (S_{s\lhd} \times Act \times S_{s\lhd})$ and
- $Adm_{s\lhd} = Adm \cap \longrightarrow^\infty_{s\lhd}$.

For any term t in $\mathcal{T}_\epsilon$, let $\mathcal{D}_\epsilon(t) = (S_{d(t)}, (\bot, *), \longrightarrow_t, Adm_{d(t)}, Act)$. Recall that $S_{d(t)} = \{(\alpha, e) \mid \alpha \in \mathsf{Inf}$ and $e \in \lfloor\!\lfloor t \rfloor\!\rfloor(\alpha)\}$ and $*$ is the unique element of $\lfloor\!\lfloor t \rfloor\!\rfloor(\bot)$. Let $\mathcal{O}_\epsilon(t) = (S_{o(t)}, t, \rightarrow, Adm_{o(t)}, Act)$. Note that if t' is a closed term and t is a term with one free variable, say x, then $[\![t[t'/x]]\!] = [\![t]\!]([\![t']\!])$. For t a term in $\mathcal{T}_g$ and $s = (\alpha, e) \in S_{d(t)}$ define *the height of s* by $h(s) = |\alpha| \in \omega$. Note that if $h(s) = n$ then $(\bot, *) \longrightarrow^n s$.

We are now ready to define the underlying maps of states $f_t \colon S_{d(t)} \to S_{o(t)}$ for the morphisms $F_t \colon \mathcal{D}_\epsilon(t) \to \mathcal{O}_\epsilon(t)$.

Definition 7.9.3 *Let $\mathcal{S}_\mathcal{T} = \{(s, t) \mid s \in S_{d(t)}$ and $t \in \mathcal{T}_g\}$. Define $f \colon \mathcal{S}_\mathcal{T} \to \mathcal{T}_g$ by well founded recursion as follows (writing $f_t(s)$ for $f(s, t)$)*

- $f_t(\bot, *) = t$,
- $f_{a:t}(a\alpha, e) = f_t(\alpha, e)$,

[2] However, we use the convention from [84] that $\lambda + 1$ is the successor of λ.

- $f_{\Sigma_{i\in I}t_i}\big(\alpha,(\mathsf{sum}\ i,s)\big) = f_{t_i}(s)$,
- $f_{t_1\times t_2}(\alpha, s_1\times s_2) = f_{t_1}(s_1)\times f_{t_2}(s_2)$,
- $f_{\mathsf{rec}\,\mathsf{x}.t}(s) = f_{t[\mathsf{rec}\,\mathsf{x}.t/x]}\big(\mathcal{E}l(\rho_t)s\big) \quad$ *if* $h(s)>0$,
- $f_{t\ A}(s) = f_t(s)\restriction A \quad$ *if* $h(s)>0$,
- $f_{\epsilon_n t}\big(1^{n'},(\mathsf{del}\ n',(\bot,*))\big) = \epsilon_{n+n'}t$,
- $f_{\epsilon_n t}\big(1^{n'}\alpha,(\mathsf{del}\ n',s)\big) = f_t(s) \quad$ *if* $|\alpha|>0$.

where $\rho_t\colon [\![\mathsf{rec}\,\mathsf{x}.t]\!] \to [\![t]\!]([\![\mathsf{rec}\,\mathsf{x}.t]\!])$ *is the isomorphism defined in Sec.7.6.2 and the well founded order on* $\mathcal{S}_{\mathcal{T}}$ *is the lexicographical order given by* $(s_1,t_1)<(s_2,t_2)$ *if* $h(s_1)<h(s_2)$ *or* $h(s_1)=h(s_2)$ *and* $gd(t_1)<gd(t_2)$.

It is not difficult to check from the definitions in Sec. 7.6 that f_t is only applied to states in $S_{d(t)}$ on the right hand side of the defining equations above.

From the map $f\colon \mathcal{S}_{\mathcal{T}} \to \mathcal{T}_g$ we get a collection of maps $\{f_t\colon S_{d(t)} \to \mathcal{T}_g \mid t\in\mathcal{T}_g\}$ that are nicely related to each other.

Lemma 7.9.4 *Let* $\{f_t\colon S_{d(t)} \to \mathcal{T}_g \mid t\in\mathcal{T}_g\}$ *be the collection of maps given above. Then there exists a collection of isomorphisms of generalised synchronisation trees* $\{\sigma_{t,s}\colon \mathcal{D}_\epsilon(t)_{s\lhd} \to \mathcal{D}_\epsilon\big(f_t(s)\big) \mid t\in\mathcal{T}_g$ *and* $s\in S_{d(t)}\}$ *such that if* $s \longrightarrow_t^* s'$ *in* $\mathcal{D}_\epsilon(t)$ *then*

$$(f_t(s)=t') \Rightarrow f_t(s') = f_{t'}\big(\sigma_{t,s}(s')\big), \tag{7.17}$$

Proof. (Sketch) We proceed by induction in the height of the states s. First we define $\sigma_{t,s}\colon \mathcal{D}_\epsilon(t)_{s\lhd} \to \mathcal{D}_\epsilon\big(f_t(s)\big)$ for $t\in\mathcal{T}_g$ and $s=(\bot,*)\in S_{d(t)}$, i.e. for all roots. Then $\mathcal{D}_\epsilon(t)_{s\lhd} = \mathcal{D}_\epsilon(t)$ and $f_t(s)=t$ so we can define $\sigma_{t,s}=1_{\mathcal{D}_\epsilon(t)}$. We then define $\sigma_{t,s}\colon \mathcal{D}_\epsilon(t)_{s\lhd} \to \mathcal{D}_\epsilon\big(f_t(s)\big)$ for $t\in\mathcal{T}_g$, $s\in S_{d(t)}$ and $h(s)=1$ by transfinite induction in $gd(t)$. For the induction step, assume $t\in\mathcal{T}_g$, $s\in S_{d(t)}$ and $h(s)=n+1$. Then there exists a unique s_n such that $s_n \longrightarrow_t s$ and $h(s_n)=n$. For $s\longrightarrow_t^* s'$ define $\sigma_{t,s}(s') = \sigma_{f_t(s_n),\sigma_{t,s_n}(s)}(\sigma_{t,s_n}(s'))$. It is not difficult to verify that this indeed defines an isomorphism from $\mathcal{D}_\epsilon(t)_{s\lhd}$to $\mathcal{D}_\epsilon\big(f_t(s)\big)$. Assuming $f_t(s)=t'$ and $f_t(s_n)=t''$ we get by induction $f_{t''}\big(\sigma_{t,s_n}(s)\big)=t'$ and $f_t(s') = f_{t''}\big(\sigma_{t,s_n}(s')\big) = f_{t'}\big(\sigma_{t'',\sigma_{t,s_n}(s)}(\sigma_{t,s_n}(s'))\big) = f_{t'}\big(\sigma_{t,s}(s')\big)$. □

From the lemma below it follows that the maps just defined are the underlying maps of Fin-open morphisms from $fin(\mathcal{D}_\epsilon(t))$ to $fin(\mathcal{O}_\epsilon(t))$.

Lemma 7.9.5 *Let* $\{f_t\colon S_{d(t)} \to \mathcal{T}_g \mid t\in\mathcal{T}_g\}$ *be the collection of maps given in Def. 7.9.3 above. If* $f_t(s_0)=t_0$ *for* $s_0\in S_{d(t)}$ *then*

$$\big(\exists s_1\in S_{d(t)}.s_0 \xrightarrow{a}_t s_1 \text{ and } f_t(s_1)=t_1\big) \text{ if and only if } t_0 \xrightarrow{a} t_1\ , \tag{7.18}$$

where $\longrightarrow$ *is the transition relation given by the operational semantics in Fig. 7.1 and Fig.7.3.*

Proof. We first show by transfinite induction in $gd(t)$ that

$$\big(\exists s_1\in S_{d(t)}.(\bot,*) \xrightarrow{a}_t s_1 \text{ and } f_t(s_1)=t_1\big) \text{ if and only if } t\xrightarrow{a} t_1\ .$$

Then (7.18) follows for $s_0\in S_{d(t)}$ and $f_t(s_0)=t_0$ by using (7.17) of Lem. 7.9.4. □

Corollary 7.9.6 *The maps f_t as given above defines for $t \in \mathcal{T}_g$ a map $f_t \colon S_{d(t)} \to S_{o(t)}$ which is the underlying map of a* Fin*-open morphism from $fin(\mathcal{D}_\epsilon(t))$ to $fin(\mathcal{O}_\epsilon(t))$.*

To show that the maps f_t define maps of *generalised* transition systems we show that they preserve admissible computations. For an infinite admissible computation ϕ of $\mathcal{D}_\epsilon(t)$ we can always find a non-empty prefix of the image of ϕ under f_t, in which all initially waiting subagents are fullfilled.

Lemma 7.9.7 *Let t be a term in $\mathcal{T}_g$ and $\phi \in Adm_{d(t)} \cap \longrightarrow^\omega$ an infinite admissible computation of $\mathcal{D}_\epsilon(t)$. Assume $\phi_n = (s_n, a_n, s_{n+1})$ for $n \in \omega$ and $f_t(s_n) = t_n$. Then there exists $n > 0$ such that*

$$\forall p \in Pos, \exists m \leq n . t_m(p) \text{ is not } waiting.$$

Proof. Easy induction in $sd(t_0)$ using Lem. 7.9.4. □

It follows by a simple mathematical induction that f_t preserves admissibility.

Lemma 7.9.8 *Let t be a term in $\mathcal{T}_g$. Then $f_{t\infty}(Adm_{d(t)}) \subseteq Adm_{o(t)}$, where $f_{t\infty}$ is the extension of f_t to computations as given in Def. 7.3.4.*

We can now conclude from Lem. 7.3.5, Cor. 7.9.6 and Lem. 7.9.8 that f_t defines a morphism of generalised transition systems.

Proposition 7.9.9 *Let t be a term in $\mathcal{T}_g$. Then $f_t \colon S_{o(t)} \to S_{d(t)}$ is the underlying map of states of a morphism of generalised transition systems which we will refer to as $F_t \colon \mathcal{D}_\epsilon(t) \to \mathcal{O}_\epsilon(t)$.*

To show that $F_t \colon \mathcal{D}_\epsilon(t) \to \mathcal{O}_\epsilon(t)$ is an Inf-*open* morphism we neet to check the two zig-zag conditions of Prop. 7.4.1 in Sec. 7.4. As already mentioned above, the first condition follows directly from Lem. 7.9.5. To show the second condition, it suffices to show that $f_t \colon S_{o(t)} \to S_{d(t)}$ *reflects* admissible computations, i.e. that $Adm_{f\,d(t)} \subseteq Adm_{d(t)}$, where $Adm_{f\,d(t)} = f_{t\infty}^{-1}(Adm_{o(t)}) = \{\phi \in Comp(\mathcal{D}_\epsilon(t)) \mid f_{t\infty}(\phi) \in Adm_{o(t)}\}$. The proof goes by structural induction in t and for the case $t = \mathsf{rec}\,\mathsf{x}.t'$ we will add a term $\top$ to the calculus SCCSϵ. The operational semantics is extended by adding the rule

$$\frac{}{\top \xrightarrow{a} \top} \quad (a \in Act).$$

As denotation of $\top$ we take the explicit terminal element of $\mathbf{Sp}(\widehat{\mathsf{Inf}})$, i.e. $[\![\top]\!]\alpha = \{*\}$. The map $f_\top \colon S_{d(\top)} \to S_{o(\top)}$ and isos $\sigma_{\top,s} \colon \mathcal{D}_\epsilon(\top)_{s\lhd} \to \mathcal{D}_\epsilon(f_\top(s))$ for $s \in S_{d(\top)}$ extending Def. 7.9.3 and Lem. 7.9.4 are defined in the obvious way, i.e. $f_\top(s) = \top$ for all $s \in S_{o(\top)}$ and $\sigma_{\top,(\alpha,*)}(\alpha\alpha', *) = (\alpha', *)$. We then use the following property of the maps f_t in connection with substitution.

Lemma 7.9.10 *Let t be a term of $\mathcal{T}_g^o$ such that $FV(t) = \{x\}$. If $m\colon [\![t']\!] \to [\![t'']\!]$ is a morphism such that*

$$\forall s \in S_{d(t')}, \forall p \in Pos, \forall n > 1$$
$$\big(\exists u''.f_{t''}(\mathcal{E}l(m)s)p = \epsilon_n u'' \Rightarrow \exists u'.f_{t'}(s)p = \epsilon_n u'\big)$$

then

$$\forall s \in S_{d(t[t'/x])}, \forall p \in Pos, \forall n > 1$$
$$\big(\exists u''.f_{t[t''/x]}(\mathcal{E}l([\![t]\!]m)s)p = \epsilon_n u'' \Rightarrow \exists u'.f_{t[t'/x]}(s)p = \epsilon_n u'\big)$$

Proof. Assume that $m\colon [\![t']\!] \to [\![t'']\!]$ is a morphism such that

$$\forall s \in S_{d(t')}, \forall p \in Pos, \forall n > 1$$
$$\big(\exists u''.f_{t''}(\mathcal{E}l(m)s)p = \epsilon_n u'' \Rightarrow \exists u'.f_{t'}(s)p = \epsilon_n u'\big)$$

By well founded induction we prove for $s \in S_{d(t[t'/x])}$ and $t \in \mathcal{T}_g^o$ with $FV(t) = \{x\}$ that

$$\forall p \in Pos, \forall n > 1 \big(\exists u''.f_{t[t''/x]}(\mathcal{E}l([\![t]\!]m)s)p = \epsilon_n u'' \Rightarrow \exists u'.f_{t[t'/x]}(s)p = \epsilon_n u'\big)$$

The well founded order is, as in Def. 7.9.3, given by $(s_1, t_1) < (s_2, t_2)$ if $h(s_1) < h(s_2)$ or $h(s_1) = h(s_2)$ and $gd(t_1) < gd(t_2)$. □

We only use the lemma in two special cases, giving the two corollaries below.

Corollary 7.9.11 *Let t be a term in $\mathcal{T}_g^o$ such that $FV(t) = \{x\}$ and $m\colon [\![t']\!] \to [\![\top]\!]$ the unique morphism into the terminal presheaf. Then*

$$\forall \phi \in Comp(\mathcal{O}_\epsilon(t[t'/x])),$$
$$f_{t[\top/x]_\infty}(\mathcal{E}l([\![t]\!]m)_\infty\phi) \text{ is inadmissible } \Rightarrow f_{t[t'/x]_\infty}(\phi) \text{ is inadmissible.}$$

For t a standard term in $\mathcal{T}_g^o$ such that $FV(t) = \{x\}$ we define $t^0 = x$ and $t^{n+1} = t^n[t/x]$.

Corollary 7.9.12 *Let t be a standard term in $\mathcal{T}_g^o$ such that $FV(t) = \{x\}$ and let $\rho_t\colon [\![\mathsf{rec}\,\mathsf{x}.t]\!] \to [\![t[\mathsf{rec}\,\mathsf{x}.t/x]]\!]$ be the isomorphism given in Sec. 7.6.2. Then $\forall n \in \omega, \forall \phi \in Comp(\mathcal{O}_\epsilon(t^n[\mathsf{rec}\,\mathsf{x}.t/x]))$,*

$$f_{t^{n+1}[\mathsf{rec}\,\mathsf{x}.t/x]_\infty}(\mathcal{E}l([\![t]\!]^n\rho_{t_1})_\infty\phi) \text{ is inadmissible } \Rightarrow f_{t^n[\mathsf{rec}\,\mathsf{x}.t/x]_\infty}(\phi) \text{ is inadmissible.}$$

Proof. By definition $f_{\mathsf{rec}\,\mathsf{x}.t}(s) = f_{t[\mathsf{rec}\,\mathsf{x}.t/x]}\big(\mathcal{E}l(\rho_t)s\big)$ if $h(s) > 0$ and since t is a standard term we have $\forall p \in Pos, f_{t[\mathsf{rec}\,\mathsf{x}.t/x]}(\bot, *)p = \epsilon_n u \Rightarrow n = 0$, so we get that $\forall s \in S_{d(t[\mathsf{rec}\,\mathsf{x}.t/x])} \forall p \in Pos \forall n > 1, f_{t[\mathsf{rec}\,\mathsf{x}.t/x]}\big(\mathcal{E}l(\rho_t)s\big)p = \epsilon_n u \Rightarrow f_{\mathsf{rec}\,\mathsf{x}.t}(s)p = \epsilon_n u$ and the desired result follows from Lem. 7.9.10 and Def. 7.5.2, by noting that $t^{n+1}[\mathsf{rec}\,\mathsf{x}.t/x] = t^n[t[\mathsf{rec}\,\mathsf{x}.t/x]/x]$ and $[\![t]\!]^n = [\![t^n]\!]$. □

Lemma 7.9.13 *Let t be a term in $\mathcal{T}_g^o$ such that $FV(t) = \{x\}$. Then*

$\forall t' \in \mathcal{T}_g \cup \{\top\}, Adm_{f_{d(t')}} \subseteq Adm_{d(t')} \Rightarrow Adm_{f_{d(t[t'/x])}} \subseteq Adm_{d(t[t'/x])}$

implies

$\forall t' \in \mathcal{T}_g \cup \{\top\}, \forall n \in \omega,$

$$Adm_{f_{d(t')}} \subseteq Adm_{d(t')} \Rightarrow Adm_{f_{d(t^n[t'/x])}} \subseteq Adm_{d(t^n[t'/x])}$$

Proof. By an easy induction in n. □

Proposition 7.9.14 *Let t be a standard term in $\mathcal{T}_g^o$ such that $FV(t) \subseteq \{x\}$. Then for all t' in $\mathcal{T}_g \cup \{\top\}$, $Adm_{f_{d(t')}} \subseteq Adm_{d(t')}$ implies $Adm_{f_{d(t[t'/x])}} \subseteq Adm_{d(t[t'/x])}$.*

Proof. (Sketch) By structural induction in t, using Lem. 7.9.12, Lem. 7.9.11 and Lem 7.9.13 above in the case for recursion. □

If we take t to be a closed term in the proposition above and e.g. $t' = \top$ then $t[t'/x]) = t$ so we get that f_{t_∞} reflects admissibility, which was what we wanted to show.

Corollary 7.9.15 *Let t be a standard term in $\mathcal{T}_g$. Then $Adm_{f_{d(t)}} \subseteq Adm_{d(t)}$.*

Chapter 8

A Relational Model of Non-Deterministic Dataflow

Abstract: We recast dataflow in a modern categorical light using profunctors as a generalisation of relations. The well known causal anomalies associated with relational semantics of indeterminate dataflow are avoided, but still we preserve much of the intuitions of a relational model. The development fits with the view of categories of models for concurrency and the general treatment of bisimulation they provide. In particular it fits with the recent categorical formulation of feedback using traced monoidal categories. The payoffs are: (1) explicit relations to existing models and semantics, especially the usual axioms of monotone IO automata are read off from the definition of profunctors, (2) a new definition of bisimulation for dataflow, the proof of the congruence of which benefits from the preservation properties associated with open maps and (3) a treatment of higher-order dataflow as a biproduct, essentially by following the geometry of interaction programme.

Introduction

A fundamental dichotomy in concurrency is the distinction between *asynchronous* communication and *synchronous* communication. In the present paper we unify the analysis of these situations in the framework of a categorical presentation of models for concurrency as initiated by Winskel and Nielsen [146]. In particular we have given a treatment of indeterminate dataflow networks in terms of (a special kind of) profunctors which is very close to the treatment of synchronous communication. This new semantical treatment has a number of benefits

1. the general functoriality and naturality properties of presheaves *automatically* imply the usually postulated axioms for asynchronous, monotone automata [103, 120]
2. we get a notion of bisimulation, which is crucial when one includes both synchronous and asynchronous primitives together,
3. it is closely connected to the extant models [68] expressed in terms of trace sets, but also provides a relational viewpoint which allows one to think of composing network components as a (kind of) relational composition,
4. gives a semantics of higher-order networks almost for "free" by using the passage from traced monoidal categories to compact-closed categories [3, 70] (the "geometry of interaction" construction).

The categorical presentation is critical for all these points. Without the realization that Kahn processes can be described as a traced monoidal category and knowledge of the results in [3, 70] it would be hard to see how one could have proposed our model of higher-order processes. It is notable that the profunctor semantics of dataflow yields automatically the axioms for monotone port automata used in modeling dataflow [103] in contrast to the work in [124]. At the same time we have to work to get a correct operation on profunctors to model the dataflow feedback; "the obvious" choice of modeling feedback by coend doesn't account for the subtle causal constraints which plague dataflow semantics.

The background for our paper includes work done on presenting models for concurrency as categories, as summarised in [146]. This enabled a sweeping definition of bisimulation based on open maps applicable to any category of models equipped with a distinguished subcategory of paths [71]. It also exposed a new space of models: presheaves. Presheaf categories possess a canonical choice of open maps and bisimulation, and can themselves be related in the bicategory of profunctors. This yields a form of domain theory but boosted to the level of using categories rather than partial orders as the appropriate domains.

One argument for the definition of bisimulation based on open maps is the powerful preservation properties associated with it. Notable is the result of [23] that any colimit preserving functor between presheaf categories preserves bisimulation, which besides obvious uses in relating semantics in different models with different notions of bisimulation is, along with several other general results, useful in establishing congruence properties of process languages. By understanding dataflow in terms of profunctors we are able to exploit the framework not just to give a definition of bisimulation between dataflow networks but also in showing it to be a congruence with respect to the standard operations of dataflow.

A difficulty has been in understanding the operational significance of the bisimulation which comes from open maps for higher-order process languages (where for example processes themselves can be passed as values). Another gap, more open and so more difficult

to approach, is that whereas both interleaving models and independence models like event structures can be recast as presheaf models, as soon as higher-order features appear, the presheaf semantics at present reduce concurrency to nondeterministic interleaving. A study of nondeterministic dataflow is helpful here as its compositional models are forced to account for causal dependency using ideas familiar from independence models; at the same time the models are a step towards understanding higher-order as they represent nondeterministic functions from input to output.

The idea that non-deterministic dataflow can be modeled by some kind of generalised relations fits with that of others, notably Stark in [124, 126]. Bisimulation for dataflow is studied in [127]. That dataflow should fit within a categorical account of feedback accords for instance with [76, 3]. But in presenting a semantics of dataflow as profunctors we obtain the benefits to be had from placing nondeterministic dataflow centrally within categories of models for concurrency, and in particular within presheaf models. One of our future aims is a dataflow semantics of hardware-description languages, like for instance Verilog HDL [49], which presently only possesses a non-compositional, operational definition. The semantics of a language of this richness requires a flexible yet abstract domain theory of the kind presheaf models seem able to support.

8.1 Models for Indeterminate Dataflow

The Dataflow paradigm for parallel computation, originated in work of Jack Dennis and others in the mid-sixties [74, 33]. The essential idea is that data flows between asynchronous computing agents, that are interconnected by communication channels acting as unbounded buffers. Traditionally, the *observable behaviour* is taken to be the *input-output* relation between sequences of values on respectively input and output channels, sometimes referred to as the *history model* [68]. For dataflow networks built from only *deterministic* nodes, Kahn [74] has observed that their behaviour could be captured *denotationally* in the history model, defining network composition by the least fixed point of a set of equations describing the components, which was later shown formally by several authors, e.g. Faustini [36], Lynch and Stark [81]. Subsequently, different semantics have been described as satisfying *Kahn's principle* when they are built up compositionally along similar lines.

8.1.1 The Need for Causality

For *indeterminate* networks, the situation is not so simple. Brock and Ackerman[17] showed the fact, referred to as the "Brock-Ackerman anomaly", that for networks containing the nondeterministic primitive *fair merge*, the history model preserves too little information about the structure of the networks to support a compositional semantics. Later, Trakhtenbrot and Rabinovich [112, 113, 114], and independently, Russell [116] gave examples of anomalies showing that this is true even for the simplest nondeterministic primitive[1] the ordinary *bounded choice*. We present a similar example to illustrate what additional information is needed. It works by giving two simple examples of automata $\mathcal{A}_1$ and $\mathcal{A}_2$, which have the same input-output relation, and a context $\mathcal{C}[-]$ as shown in the figure, in which they behave differently. The context consists of a fork process $\mathcal{F}$ (a process that copies every input to two outputs), through which the output of the automata $\mathcal{A}_i$ is fed back to the input channel.

[1] See [103, 102] for a study of the differences between the nondeterminate primitives.

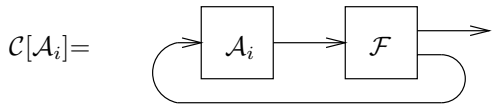


Figure 8.1: The automata $\mathcal{A}_i$ inserted in context $\mathcal{C}[-]$ consisting of a fork process $\mathcal{F}$ and a feedback loop

Automaton $\mathcal{A}_1$ has the following (deterministic) behaviour: It outputs a token; waits for a token on input and then outputs another token. Automaton $\mathcal{A}_2$ has the choice between two behaviours: Either it outputs a token and stops, *or* it waits for an input token, then outputs two tokens. For both automata, the IO-relation relates empty input to zero or one output token, and non-empty input to zero, one or two output tokens, but $\mathcal{C}[\mathcal{A}_1]$ can output two tokens, whereas $\mathcal{C}[\mathcal{A}_2]$ can only output a single token, choosing the first behaviour of $\mathcal{A}_2$. This example shows the need for a model that records a more detailed causality relation between individual data tokens than the history model.

Jonsson [68] and Kok [78] have independently given fully abstract models for indeterminate dataflow. Jonsson's model is based on trace[2] sets, which are sets of possible interaction sequences, finite or infinite, between a process and its environment. Kok's model turned out to be equivalent. Rabinovich and Traktenbrot analyzed the same issues from the point of view of finite observations and came up with general conditions under which a Kahn-like principle would hold [112, 113, 114]. Abramsky has generalised Kahn's principle to indeterminate networks [2].

8.2 A Traced Monoidal Category of Kahn Processes

In this section we summarize the basic theory of traced monoidal categories and describe a category of Kahn processes as an instance of a traced monoidal category. The notion of traced monoidal category abstracts the notion of trace of a matrix from multi-linear algebra. However it has emerged in a variety of new contexts including the study of feedback systems [9], knot theory [67] and recursion [52]. The axiomatization presented below is the definition of Joyal, Street and Verity [70], slightly simplified and specialized as in [52] to the context of (strict) symmetric monoidal categories so that the axioms appear simpler; in particular we do not consider braiding or twists. In the Joyal, Street and Verity paper the fact that trace models feedback (or iteration) is attributed to Bloom, but as far back as 25 years ago Bainbridge had been studying trace in the context of feedback in systems and control theory. Indeed Bainbridge had noticed that there were two kinds of trace (associated with two different monoidal structures) in Rel, the category of sets and binary relations. Furthermore he noted that one of the traces corresponds to feedback in what are essentially memoryless Kahn networks. [3]

[2]This word commonly used in the literature unfortunately clashes with "trace" in linear algebra. Normally this is not a problem but the present paper uses this word in both senses, we hope the reader will be able to disambiguate from the context.

[3]We are indebted to Samson Abramsky for pointing this reference out to us.

8.2.1 Traced Monoidal Categories

In this section we give the axioms for a strict symmetric monoidal category equipped with a trace. We assume that the reader is familiar with the notion of a (strict) symmetric tensor product. We write $\otimes$ for the tensor product and $\sigma_{XY} : X \otimes Y \to Y \otimes X$ for the natural isomorphism (the symmetry) in this case.

Definition 8.2.1 *A **trace** for a symmetric monoidal category $\mathcal{C}$ is a family of functions*

$$\mathsf{Tr}^U_{X,Y}() : \mathcal{C}(X \otimes U, Y \otimes U) \to \mathcal{C}(X, Y)$$

satisfying the following conditions

1. Bekic*: $f\colon X \otimes U \otimes V \to Y \otimes U \otimes V$ and $g\colon X \to Y$*

$$\mathsf{Tr}^{U\otimes V}_{X,Y}(f) = \mathsf{Tr}^U_{X,Y}(\mathsf{Tr}^V_{X\otimes U, Y\otimes U}(f)) \quad \textit{and } \mathsf{Tr}^I_{X,Y}(g) = g \ .$$

2. Yanking*: $\mathsf{Tr}^U_{U,U}(\sigma_{UU}) = I_U$.*

3. Superposing*: Given $f\colon X \otimes U \to Y \otimes U$*

$$\mathsf{Tr}^U_{Z\otimes X, Z\otimes Y}(I_Z \otimes f) = I_Z \otimes \mathsf{Tr}^U_{X,Y}(f) \ .$$

4. Naturality*: Given $g\colon Z \otimes U \to Y \otimes U$, $f\colon X \to Z$ and $h\colon Y \to W$*

$$\mathsf{Tr}^U_{X,W}((h \otimes I_U) \circ g \circ (f \otimes I_U)) = h \circ \mathsf{Tr}^U_{Z,Y}(g) \circ f \ .$$

5. Dinaturality*: Given $f\colon X \otimes U \to Y \otimes V$ and $g\colon V \to U$*

$$\mathsf{Tr}^U_{X,Y}((I_Y \otimes g) \circ f) = \mathsf{Tr}^V_{X,Y}(f \circ (I_X \otimes g)) \ .$$

The intuition of a traced symmetric monoidal category as a category with *feedback* becomes clear when the axioms are presented graphically as in Fig. 8.2. The symmetry is indicated by a cross inside the box, swapping the channels and identities just as straight arrows.

The following proposition is an easy consequence of the yanking and naturality conditions, keeping in mind functoriality of $\otimes$ and naturality of symmetries. It shows how composition can be defined from trace and tensor as illustrated by Fig. 8.3.

Proposition 8.2.2 *Given $g\colon U \to Y$ and $f\colon X \to U$ we have*

$$\mathsf{Tr}^U_{X,Y}(\sigma_{UY} \circ (f \otimes g)) = g \circ f \ .$$

This could be viewed as a generalisation of the yanking condition.

It is instructive to consider the two well-known examples of trace in Rel. In the first case one takes the tensor product to be the cartesian product of the underlying sets and in the second case one takes the tensor product to be disjoint union of sets (with the evident action on relations); we call these structures $(\mathsf{Rel}, \times)$ and $(\mathsf{Rel}, +)$ respectively. The trace in $(\mathsf{Rel}, \times)$ is given by

$$\mathsf{Tr}^U_{X,Y}(R)(x, y) = \exists u \in U.R(x, u, y, u) \ ,$$

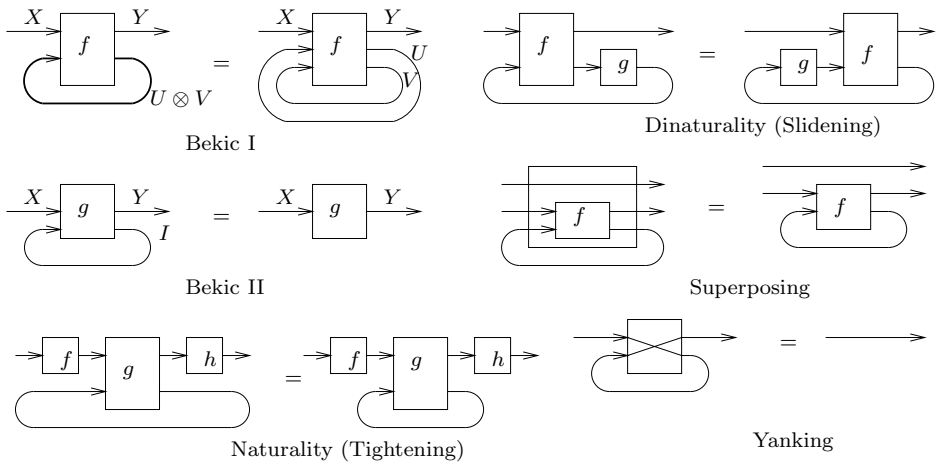


Figure 8.2: Graphical presentation of the axioms for a traced strict symmetric monoidal category

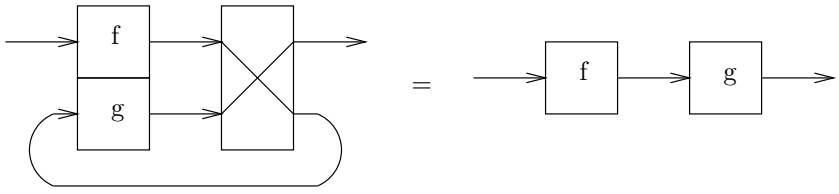


Figure 8.3: The generalised Yanking property

where R is a binary relation from $X \times U$ to $Y \times U$. This is very close to the trace in linear algebra - the sum along the diagonal - here it is an existential quantification along the "diagonal." Now for the other structure one proceeds as follows. Let R be a binary relation from $X \uplus U$ to $Y \uplus U$. This can be seen as consisting of 4 pieces, namely the relations R_{XY}, R_{XU}, R_{UY} and R_{UU}. For example we say that $R_{XY}(x, y)$ holds for $x \in X, y \in Y$ iff $R(x, y)$ holds. Now the trace is given by

$$\mathsf{Tr}^U_{X,Y}(R) = R_{XY} \cup R_{XU}; R^*_{UU}; R_{UY} \ ,$$

where we are using the standard relational algebra concepts; $*$ for reflexive, transitive closure, ; for relational composition and $\cup$ for union of the sets of pairs in the relation. Intuitively this is the formula expressing feedback: either x and y are directly related or x is related to some u and that u is related to y (once around the feedback loop) or, more generally, we can go around the "feedback loop" an indefinite number of times.

8.2.2 The Kahn Category

The basic intuitions behind Kahn networks are, of course, due to Kahn [74] and a formal operational semantics in terms of coroutines is due to Kahn and McQueen [73]. The particular axiomatisation presented here builds on the ideas of Stark [124] but using the formalism of traces presented in [102]. No originality is claimed for the trace model, it was Bengt Jonsson [68] who showed that traces form a fully abstract model of dataflow networks and there were several others with similar ideas at the time.

We have a fixed set $\mathcal{V}$ of *values*. A dataflow network then processes values on a finite set of input ports producing values on a finite set of output ports. Following [127](Stark: Dataflow calculus) we will not assume a fixed set of port names as in [], but simply refer to the nth port by the number $n \in \omega$. By doing this we need not consider questions as e.g. name clashes and can work with the simpler categorical structures of *strict* monoidal categories. An *event* is a triple $\langle a, i/o, v\rangle$ where $a \in \omega$ (the number of the channel) and $v \in \mathcal{V}$. We say that $\langle a, v\rangle$ is the *label* of the event $\langle a, i/o, v\rangle$. An event of the form $\langle a, o, v\rangle$ is called an *output* event and one of the form $\langle a, i, v\rangle$ is called an *input* event. We consider sequences of these events. If α is a sequence of events we write $l(\alpha)$ for the sequence of labels obtained by discarding the input/output tags. We write $\alpha|_o$ (or $\alpha|_i$) for the sequence of output (or input) events discarding the input (or output) events. For $n \in \omega$ we write $\alpha|^{<n}$ for the sequence obtained from α by keeping only the input events on the ports below n. We write $\alpha|^{\geq n}$ for the sequence obtained from α by keeping the input events on the ports higher or equal n, subsequently renumbered by subtracting n from the port number. We define $\alpha|_{<n}$ and $\alpha|_{\geq n}$ similarly just for output events. Finally, we allow all combinations of these restrictions, e.g. $\alpha|^{<n}_{\geq m}$ is the sequence obtained by keeping the input events on ports below n and output events on ports above or equal m, and subsequently renumbering the ports of the output events by subtracting m from the port number.

We extend these notations to sets of sequences. For $m \leq n \in \omega$ let $[m \ldots n] = \{m, m + 1, \ldots, n - 1\}$ and $[n] = [0 \ldots n]$. We write $\mathcal{I}_n$ for the set $[n] \times \{i\} \times \mathcal{V}$ of all input events on ports below n and similarly $\mathcal{O}_n$ for $[n] \times \{o\} \times \mathcal{V}$. Finally, we write $\mathcal{L}_n$ for the set $[n] \times \mathcal{V}$ of labels on ports below n.

Definition 8.2.3 *A **process** of **sort** (n, m), is a non-empty prefix closed set of finite sequences over the alphabet $\mathcal{I}_n \cup \mathcal{O}_m$. The set of sequences, say S, satisfies the following closure*

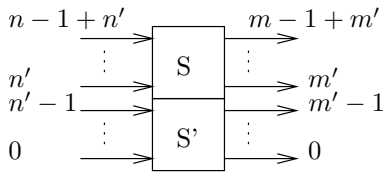


Figure 8.4: The shuffle of processes $S\colon n \to m$ and $S'\colon n' \to m'$.

properties, α and β are sequences of events:

K1. *If $\alpha\langle b,o,v\rangle\langle a,i,u\rangle\beta \in S$ then $\alpha\langle a,i,u\rangle\langle b,o,v\rangle\beta \in S$.*

K2. *If $\alpha\langle b,o,v\rangle\langle b',o,u\rangle\beta \in S$ and if $b \neq b'$ then $\alpha\langle b',o,u\rangle\langle b,o,v\rangle\beta \in S$.*

K3. *If $\alpha\langle a,i,u\rangle\langle a',i,v\rangle\beta \in S$ and if $a \neq a'$ then $\alpha\langle a',i,v\rangle\langle a,i,u\rangle\beta \in S$.*

K4. *If $\alpha \in S$ then $\alpha\langle a,i,v\rangle$ for all $\langle a,v\rangle \in \mathcal{L}_A$.*

We say that n is the input arity *of S and m is the* output arity *of S.*

The last condition above is called *receptivity*, a process could receive any data on its input ports; unlike with synchronous processes. Receptivity is the basic reason why traces suffice to give a fully-abstract model for asynchronous processes; in calculi with synchronous communication one needs branching information.

The first three conditions express concurrency conditions on events occurring at different ports. Note an asymmetry in the first condition. If an output occurs before an input then it could also occur after the input instead. However, if an output occurs after an input then the pair of events cannot be permuted because the output event may be in response to the input. Furthermore we are assuming, again in (1), that the arrival of input does not disable already enabled output. In an earlier investigation [103] these were called *monotone* automata and it was shown that many common primitives, such as fair merge, timeouts, interrupts and polling cannot be expressed as monotone automata. However, as we will see in the end of this section, monotonicity and receptivity together imply that the IO-relations of Kahn processes are *buffered* in a formal sense. This makes them reasonable assumptions for the type of networks we consider. The restriction to finite sequences is a simplification, which is not necessary for the results in this section, but simplifies the exposition in the following section. In the model of Jonsson infinite sequences are used to express fairness properties.

Given processes as sets of sequences we define a strict symmetric monoidal category of Kahn processes. We write $S\colon n \to m$ for "S is a process of sort (n,m)". We begin by defining the shuffle of two processes, which corresponds to putting the two processes next to each other as illustrated in Fig. 8.4.

Definition 8.2.4 *Given two processes $S\colon n \to m$ and $S'\colon n' \to m'$. We define the set $S\Delta S'\colon (n+n') \to (m+m')$ (read, S **shuffle** S') as the set of all sequences γ of sort $(n+n', m+m')$ s.t. $\gamma|^{<n'}_{<m'} \in S'$ and $\gamma|^{\geq n'}_{\geq m'} \in S$.*

We then define composition of two processes with matching output and input arity to be the shuffle of the two processes, from which we have picked the sequences with the right causal precedence of events on the connected ports and then discarded these, now "internal", events.

Definition 8.2.5 *Given processes $f\colon n \to m$ and $g\colon m \to p$ we define the composite $f;g\colon n \to p$ of f and g by $f;g = S|^{\geq m}_{\leq p}$, where $S \subseteq f\Delta g$ is the largest set s.t. $\forall \delta \in S.\ l(\delta|^{<m}) = l(\delta|_{\geq p})$ & $\forall \delta' \leq \delta.\ l(\delta'|^{<m}) \leq l(\delta'|_{\geq p})$.*

Definition 8.2.6 *The category* Kahn *of Kahn processes has as objects natural numbers and as morphisms from n to m, processes of sort (n, m) Composition of morphisms is defined by composition of processes as defined above. For $n \in \omega$, the identity on n is given by $1_n = \{\alpha \in (\mathcal{I}_n \cup \mathcal{O}_n)^* \mid \forall \alpha' \leq \alpha,\ l(\alpha'|_o) \leq l(\alpha'|_i)\}$.*

A strict monoidal structure is given by sum on objects and for $f\colon n \to m$ and $f'\colon n' \to m'$, $f \otimes f'\colon n + n' \to m + m'$ is given by $f\Delta g$. The trace construction is as follows. Given $f\colon n + p \to m + p$ we define $Tr^p_{n,m}(f)\colon n \to m$ as the set of all sequences γ such that there is a sequence $\delta \in f$ with

1. $\delta|^{\geq p}_{\geq p} = \gamma$,
2. $l(\delta|^{<p}) = l(\delta|_{<p})$ and
3. $\forall \delta' \leq \delta.\ l(\delta'|^{<p}) \leq l(\delta'|_{<p})$,

i.e. for all $0 \leq i < p$, the values on output channel i are fed back into input channel i.

Theorem 8.2.7 *With the structures given above,* Kahn *is a traced (strict) symmetric monoidal category.*

The generalised yanking property can be interpreted in this category as saying that composition can be obtained as a combination of parallel composition (that is, shuffling) and feedback. This is a well-known fact in dataflow folklore.

8.2.3 From Kahn Processes to Input-output Relations

The category of Kahn processes can be related to the history model by a functor to a category of buffered relations between histories. Given a set of port indices $[n]$ for $n \in \omega$, let H_n, the *histories* on $[n]$, be the elements in the free partially commutative monoid $\mathcal{L}^*_n/\eqsim$ [34], where $\eqsim$ is the smallest equivalence relation such that $\alpha\langle a', v'\rangle\langle a, v\rangle\beta \eqsim \alpha\langle a, v\rangle\langle a', v'\rangle\beta$ if $a \neq a'$. This is an example of a *Mazurkiewicz trace language*, why we will also refer to histories as *traces*. For a sequence $\alpha \in \mathcal{L}^*_n$ let $\overline{\alpha}$ denote its trace. A trace $\overline{\alpha}$ can also be viewed as a function $\overline{\alpha}\colon [n] \to \mathcal{V}^*$ which is the traditional representation of histories [68]. An *IO-relation* R with input arity n and output arity m is a subset $R \subseteq H_n \times H_m$.

The traces in H_n can be partial ordered by $\overline{\alpha} \sqsubseteq \overline{\beta} \in \mathcal{L}^*_n/\eqsim$ iff $\exists \gamma \in \mathcal{L}^*_n$ such that $\overline{\alpha\gamma} = \overline{\beta}$ (see Sect.7 of [146]). Viewed as functions the ordering is just the point-wise prefix ordering. Let ϵ_n (or just ϵ if the arity n is clear from the context) denote the empty trace in H_n. Given the ordering on traces, we can define a *buffer* $B_n \subseteq H_n \times H_n$ for each arity $n \in \omega$, as the relation $\{(\overline{\alpha}, \overline{\beta}) \in H_n \times H_n \mid \overline{\beta} \sqsubseteq \overline{\alpha}\}$. Inspired by the work in [120], we say that an IO-relation $R \subseteq H_n \times H_m$ is *buffered* if it satisfies that $R = B_n; R; B_m$, i.e. adding buffers to input and output makes no difference. The lemma below gives a characterisation of buffered IO-relations.

Lemma 8.2.8 *Let $R \subseteq H_n \times H_m$. Then the relation R is buffered if and only if*

- *If* $(\overline{\alpha}, \overline{\beta}) \in R$ *and* $\overline{\alpha} \sqsubseteq \overline{\alpha}'$ *then* $(\overline{\alpha}', \overline{\beta}) \in R$.
- *If* $(\overline{\alpha}, \overline{\beta}) \in R$ *and* $\overline{\beta}' \sqsubseteq \overline{\beta}$ *then* $(\overline{\alpha}, \overline{\beta}') \in R$.

Proof. Straightforward. □

From the fact that $B_n; B_n = B_n$ for any $n \in \omega$ it follows that we can define a category Hist of buffered IO-relations, in which the identities are given by the buffers.

Definition 8.2.9 *Let* Hist *be the category with objects* H_n *for* $n \in \omega$ *and morphisms being buffered IO-relations.*

We can define a symmetric monoidal structure as in $(\mathsf{Rel}, \times)$. Using the immediate isomorphism $H_{n+m} \cong H_n \times H_m$, in one direction mapping $\overline{\delta}$ in H_{n+m} to $(\overline{\delta}|^{\geq m}_{\geq m}, \overline{\delta}|^{\leq m}_{<m})$, we define a *strict* symmetric monoidal structure with tensor on objects given by $H_n \otimes H_m = H_{n+m}$. In the remaining part of this paper we will often implicitly use this isomorphism, writing $(\overline{\alpha}, \overline{\beta})$ for the history $\overline{\delta}$ in $H_n \otimes H_m$ such that $\overline{\alpha} = \overline{\delta}|^{\geq m}_{\geq m}$ and $\overline{\beta} = \overline{\delta}|^{\leq m}_{<m}$.

A Kahn process S defines naturally a relation between input and output *sequences* by considering the set $\mathcal{H}(S) = \{\, (\gamma, \delta) \mid \exists \alpha \in S.\ \gamma = l(\alpha|_i)\ \&\ \delta = l(\alpha|_o) \}$. It can be shown that $\mathcal{H}$ extends to a buffered relation between traces and preserves composition.

Proposition 8.2.10 *There is a symmetric monoidal functor* $\mathcal{H} \colon \mathsf{Kahn} \to \mathsf{Hist}$ *that maps arities* n *to* H_n *and a Kahn process* S *of sort* (n, m) *to the relation* $\{\, (\overline{\gamma}, \overline{\delta}) \mid \exists \alpha \in S.\ \gamma = l(\alpha|_i)\ \&\ \delta = l(\alpha|_o) \}$.

Proof. We must show that for all $n \in \omega$, $\mathcal{H}(1_n) = B_n$ and that $\mathcal{H}(f; g) = \mathcal{H}(f); \mathcal{H}(g)$ for $f \colon n \to m$ and $g \colon m \to p$. The first part follows almost immediately from the definition of identities in the two categories. For the second, we will only show that $\mathcal{H}(f; g) \subseteq \mathcal{H}(f); \mathcal{H}(g)$, the other direction follows by reversing all implications. Assume $(\overline{\gamma}, \overline{\delta}) \in \mathcal{H}(f; g)$. Then there exists $\alpha \in f; g$ such that

$$\gamma = l(\alpha|_i)\ \&\ \delta = l(\alpha|_o). \tag{8.1}$$

By Def. 8.2.5 there exists $\phi \in f \Delta g$ such that

$$\alpha = \phi|^{\geq m}_{<p}, \tag{8.2}$$

$$l(\phi|^{<m}) = l(\phi|_{\geq p}) \qquad \text{and} \tag{8.3}$$

$$\forall \phi' \leq \phi.\ l(\phi'|^{<m}) \leq l(\phi'|_{\geq p}).$$

But this implies by Def. 8.2.4 that $\phi|^{\leq m}_{<p} \in g$ and $\phi|^{\geq m}_{\geq p} \in f$. Let $\phi_g = \phi|^{\leq m}_{<p}$, $\phi_f = \phi|^{\geq m}_{\geq p}$ and $\beta = l(\phi|^{<m})$. Now from (8.1) and (8.2) above it follows that $l(\phi_f|_i) = \gamma$ and $l(\phi_g|_o) = \delta$. From (8.3) it follows that $l(\phi_g|_i) = l(\phi_f|_o) = \beta$. But then $(\overline{\gamma}, \overline{\beta}) \in \mathcal{H}(f)$ and $(\overline{\beta}, \overline{\delta}) \in \mathcal{H}(g)$ so $(\overline{\gamma}, \overline{\delta}) \in \mathcal{H}(f; g)$. □

8.3 Generalising Relations

Kahn processes are typical of the solutions to the problem of obtaining a compositional semantics for nondeterministic dataflow. A correct compositional semantics is got by representing processes as interaction sequences, keeping track of the causal dependency between events. However, this seems far removed from the relational model. In this section we will describe another solution that *contains* the Kahn processes, which comes about as a natural (categorical) extension of the history model. Moreover, it gives a *branching* semantics, which opens the way to e.g. synchronous network primitives.

For $n \in \omega$, let $\mathsf{H_n}$ refer to the partial order category given by H_n and the ordering $\sqsubseteq$ on traces defined in the previous section. If $\overline{\alpha} \sqsubseteq \overline{\gamma}$ in H_n, let $[\overline{\alpha}, \overline{\gamma}] \colon \overline{\alpha} \to \overline{\gamma}$ and $[\overline{\gamma}, \overline{\alpha}] \colon \overline{\gamma} \to \overline{\alpha}$ denote the unique arrows in respectively $\mathsf{H_n}$ and $\mathsf{H_n}^{\mathsf{op}}$. We will refer to these categories as the *path categories*.[4]

The key observation is that buffered IO-relations between H_n and H_m correspond exactly to functors $\mathsf{H_n} \times \mathsf{H_m}^{\mathsf{op}} \to \mathbf{2}$, where $\mathbf{2}$ is the category consisting of two objects 0 and 1 and only one non-identity arrow $0 \to 1$. This is an immediate categorical analogy to characteristic functions $H_n \times H_m \to \{0, 1\}$ of relations. Viewing the relations in this way, composition of $R \colon \mathsf{H_n} \times \mathsf{H_m}^{\mathsf{op}} \to \mathbf{2}$ and $R' \colon \mathsf{H_m} \times \mathsf{H_p}^{\mathsf{op}} \to \mathbf{2}$ can be written as

$$R; R'(\overline{\alpha}, \overline{\gamma}) = \bigvee_{\overline{\beta} \in \mathsf{H_m}} R(\overline{\alpha}, \overline{\beta}) \wedge R'(\overline{\beta}, \overline{\gamma}) \ , \tag{8.4}$$

where we make use of the obvious join and meet operations on $\mathbf{2}$.

This defines a category BRel of buffered relations, with path categories as objects, arrows being relations and composition as defined above. The category BRel can be equipped with a strict symmetric monoidal structure as in Hist, and as stated below, BRel is just an alternative presentation of the category Hist given in the previous section.

Proposition 8.3.1 *The category* Hist *is (strict symmetric monoidal) equivalent to the category* BRel.

Proof. Follows easily from Lem. 8.2.8. □

A trace in BRel can be defined as in $(\mathsf{Rel}, \times)$, that is for $R \colon \mathsf{H_{n+p}} \times \mathsf{H_{m+p}}^{\mathsf{op}} \to \mathbf{2}$, define for $(\overline{\alpha}, \overline{\beta})$ in $\mathsf{H_n} \times \mathsf{H_m}^{\mathsf{op}}$,

$$\mathsf{Tr}^{\mathsf{H_p}}_{\mathsf{H_n},\mathsf{H_m}}(R)(\overline{\alpha}, \overline{\gamma}) = \bigvee_{\overline{\beta} \in \mathsf{H_p}} R((\overline{\alpha}, \overline{\beta}), (\overline{\gamma}, \overline{\beta})) \ , \tag{8.5}$$

where we have implicitly used the isomorphisms $\mathsf{H_{n+p}} \cong \mathsf{H_n} \times \mathsf{H_p}$ and $\mathsf{H_{m+p}} \cong \mathsf{H_m} \times \mathsf{H_p}$.

However, the anomaly given in Sec. 8.1.1 shows that there is no way of defining a trace on BRel such that the functor $\mathcal{H}$ given in the last section preserves the trace of Kahn. It must be possible to represent *different* dependencies between input and output for a particular input-output pair in the relation. This is precisely what moving to the bicategory of *profunctors* does for us.

[4] The traces can also be viewed as a specific kind of pomsets [110] and the path categories as a subcategory of the category of pomsets given in [71].

8.3.1 Profunctors

The (bi)category Prof of profunctors, (or bimodules, or distributors [15]) are a categorical generalisation of sets and relations. The objects of Prof are small categories and arrows are profunctors; profunctors are like the buffered relations above but with the category **2** replaced by Set.

Definition 8.3.2 *Let* P *and* Q *be small categories. A profunctor* $X\colon \mathsf{P}\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{Q}$ *is a bifunctor* $X\colon \mathsf{P}\times\mathsf{Q}^{\mathsf{op}}\to\mathsf{Set}$ *(or equivalently, a presheaf in* $\widehat{\mathsf{P}^{\mathsf{op}}\times\mathsf{Q}}$*).*

When defining profunctors from basic functors we will use sans serif letters as formal parameters, ranging over both objects and arrows. E.g. composition of profunctors $X\colon \mathsf{P}\to\mathsf{U}$ and $Y\colon \mathsf{U}\to\mathsf{Q}$ is given by the *coend* [82]

$$X;Y(\mathsf{p},\mathsf{q}) = \int^{u} X(\mathsf{p},u)\times Y(u,\mathsf{q}) \ , \tag{8.6}$$

for $Y\colon \mathsf{P}\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{U}$ and $Z\colon \mathsf{U}\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{Q}$. This defines composition only to within isomorphism, explaining why we get a *bi*category. Note how (8.6) generalises the expression for relational composition given by (8.4) earlier.

Identities $I_P\colon \mathsf{P}\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{P}$ are given by hom-functors as

$$I_P(\mathsf{p},\mathsf{p}') = \mathsf{P}[\mathsf{p}',\mathsf{p}].$$

The tensor product is given by the product of categories on objects and set-theoretic product on arrows. This defines a symmetric monoidal structure on Prof.

Definition 8.3.3 *Let* P,P' *and* Q,Q' *be small categories and* $X\colon \mathsf{P}\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{Q}$, $Y\colon \mathsf{P}'\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{Q}'$ *profunctors. Define* $\mathsf{P}\otimes\mathsf{P}' = \mathsf{P}\times\mathsf{P}'$ *and* $X\otimes Y = X\times Y\colon \mathsf{P}\otimes\mathsf{P}'\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{Q}\otimes\mathsf{Q}'$*, so* $(X\otimes Y)(\mathsf{p},\mathsf{p}',\mathsf{q},\mathsf{q}') = X(\mathsf{p},\mathsf{q})\times Y(\mathsf{p}',\mathsf{q}')$*. The symmetry* $\sigma_{PP'}\colon \mathsf{P}\otimes\mathsf{P}'\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{P}'\otimes\mathsf{P}$ *is defined in the obvious way from hom-functors, so* $\sigma_{PP'}((\mathsf{p},\mathsf{p}'),(\mathsf{q}',\mathsf{q})) = \mathsf{P}\times\mathsf{P}'[(\mathsf{q},\mathsf{q}'),(\mathsf{p},\mathsf{p}')]$.

The obvious choice of trace on Prof is to take the trace of a profunctor $X\colon \mathsf{P}\otimes\mathsf{U}\mathrel{-\!\!\!\vdash\!\!\!\mapsto}\mathsf{Q}\otimes\mathsf{U}$ to be given by

$$Tr^{U}_{P,Q}(X)(\mathsf{p},\mathsf{q}) = \int^{u} X((\mathsf{p},u),(\mathsf{q},u)) \ , \tag{8.7}$$

which satisfies the properties of a trace (up to isomorphism). In particular, we can prove the generalised yanking property.

Proposition 8.3.4 *Given* $X\colon \mathsf{P}\to\mathsf{U}$ *and* $Y\colon \mathsf{U}\to\mathsf{Q}$ *we have a (natural) isomorphism*

$$Tr^{U}_{P,Q}((X\otimes Y);\sigma_{UQ}) \cong X;Y \ .$$

Proof. *By unfolding the definitions we get*

$$\begin{aligned} Tr_P^U(Q)(X \otimes Y); \sigma_{UQ}(\mathsf{p},\mathsf{q}) &= \int^u (X \otimes Y); \sigma_{UQ}((\mathsf{p},u),(\mathsf{q},u)) \\ &= \int^u \int^{(u',q')} X \otimes Y((\mathsf{p},u),(u',q')) \times \sigma_{UQ}((u',q'),(\mathsf{q},u)) \\ &= \int^u \int^{(u',q')} X \otimes Y((\mathsf{p},u),(u',q')) \times \mathsf{U}\times\mathsf{Q}[(u,\mathsf{q}),(u',q')] \\ &= \int^u (X \otimes Y); I_{U\otimes Q}((\mathsf{p},u),(u,\mathsf{q})) \\ &\cong \int^u X(\mathsf{p},u) \times Y(u,\mathsf{q}) \ , \end{aligned}$$

where the isomorphism comes from the (natural) isomorphism in Prof *between an arrow and the arrow composed with the identity.* □

Since we are working with functors into Set, the coend in Eq. (8.7) has an explicit definition. For p and q objects of respectively P and Q, we have

$$\int^u X(p,u,q,u) \cong \bigcup\!\!\!\!\!+_{u\in\mathsf{U}} \{\, x \in X((p,u),(q,u))\}_{/\sim} \ , \tag{8.8}$$

where $\sim$ is the symmetric, transitive closure of the relation $\rightsquigarrow$ defined as follows. For $x \in X(p,u,q,u)$ and $x' \in X(p,u',q,u')$, let $x \rightsquigarrow x'$ if $\exists m\colon u \to u'$ and $y \in X(p,u,q,u')$ such that $X(p,u,q,m)y = x$ and $X(p,m,q,u')y = x'$. For $f\colon p \to p'$ and $g\colon q' \to q$ arrows of respectively P and Q, we have

$$\int^u X((f,u),(g,u))[x]_\sim = [X((f,1_u),(g,1_u))x]_\sim \quad \text{for } x \in X((p,u),(q,u)). \tag{8.9}$$

For our purpose, we focus on the subcategory PProf of Prof induced by the path categories, generalising the buffered IO-relations. A tensor product is given by $\mathsf{H_n} \otimes \mathsf{H_m} = \mathsf{H_{n+m}}$ with unit $I = \mathsf{H_0}$. Via the isomorphism $\mathsf{H_{n+m}} \cong \mathsf{H_n} \times \mathsf{H_m}$ the category PProf inherits the traced symmetric monoidal structure (up to isomorphism) of Prof. We will refer to the symmetries by $\sigma_{n,m}\colon \mathsf{H_n}\otimes\mathsf{H_m} \to \mathsf{H_m}\otimes\mathsf{H_n}$. Recall that the category $\mathsf{H_1}$ is equivalent to the category $\mathcal{V}^*$ and so a profunctor $X\colon I \to \mathsf{H_1}$ is simply a presheaf over $\mathcal{V}^*$. As shown in [71, 147], the category of (rooted) presheaves over $\mathcal{V}^*$ is equivalent to a category of synchronisation trees with label set $\mathcal{V}$. In this way, the hom-categories in PProf can be viewed as a direct generalisation of the presheaf models used in giving semantics to *synchronously* communicating systems [23], adding (asynchronous) input channels and independency between distinct channels. In the section below we will generalise the concrete representation of presheaves given in [147], which will give a more operational reading of port profunctors. However, first we will note that the trace as given by the coend fails to satisfy the causal constraints of feedback, that a token must appear as output before it appears as input on a feedback channel, as stated in the third requirement of the trace in Kahn. This is not surprising, bearing in mind the close relationship to the trace in $(\mathsf{Rel},\times)$.

Consider the fork process $\mathcal{F}\colon \mathsf{H_1} \mathrel{-\!\!\!\vdash\!\!\!\to} \mathsf{H_1}\otimes\mathsf{H_1}$ used in the example of Sect. 8.1.1, which is just a buffer copying each input to two output channels. The port profunctor corresponding

to $\mathcal{F}$ is constructed from hom-functors, on objects defined by $\mathcal{F}\big(\overline{\alpha},(\overline{\beta},\overline{\gamma})\big) = [\overline{\beta},\overline{\alpha}] \times [\overline{\gamma},\overline{\alpha}]$. Connecting one of the output channels to the input channel should result in a process with no input channels and one output channel, that can output *nothing* but the empty trace. This is indeed the result in Kahn. However, from the explicit definition of the coend given in (8.8) it is not difficult to compute that

$$\begin{aligned} Tr^{\mathsf{H}_1}_{\mathsf{H}_0,\mathsf{H}_1}(\mathcal{F})(\overline{\beta}) &\cong \biguplus_{\overline{\alpha}\in\mathsf{H}_1} \{\, x \in [\overline{\beta},\overline{\alpha}] \times [\overline{\alpha},\overline{\alpha}]\}_{/\sim} \\ &\cong \{\overline{\alpha} \mid \overline{\beta} \leq \overline{\alpha} \in \mathsf{H}_1\} \ , \end{aligned} \tag{8.10}$$

which means that $Tr^{\mathsf{H}_1}_{\mathsf{H}_0,\mathsf{H}_1}(\mathcal{F})$ can output *any* sequence.

8.3.2 An Operational Reading

In this section we shall see that it is possible to adopt the causal constraint on the trace in Kahn to the trace in port profunctors. To begin with, we will restrict attention to *rooted* profunctors $X\colon \mathsf{H_n} \mathrel{-\!\!\!\vdash\!\!\!\to} \mathsf{H_m}$, which are the profunctors satisfying that $X(\overline{\alpha},\epsilon)$ is the singleton set for any $\overline{\alpha}$ in $\mathsf{H_n}$. As mentioned above, the hom-category of rooted profunctors $\mathsf{PProf}[I,\mathsf{H}_1]$ can be shown to be equivalent to the category of synchronisation trees with label set $\mathcal{V}$ using a construction on presheaves given in [147]. For a rooted port profunctor $X\colon \mathsf{H_n} \mathrel{-\!\!\!\vdash\!\!\!\to} \mathsf{H_m}$, we define its associated *port automaton* as follows. In fact this is a special case of a (generalised) Grothendieck construction given in [128].

Definition 8.3.5 *Let $X\colon \mathsf{H_n} \mathrel{-\!\!\!\vdash\!\!\!\to} \mathsf{H_m}$ be a rooted port profunctor. Define its* associated (n,m)-port automaton $\mathcal{A}(X)$ *to be the quintuple $(S, r, \longrightarrow, [n]\ , [m]\ )$, where*

- $S = \{\big((\overline{\alpha},\overline{\beta}),x\big) \mid x \in X(\overline{\alpha},\overline{\beta})\}$ *is a set of* states,
- $r = \big((\epsilon,\epsilon),x\big)$ *for $x \in X(\epsilon,\epsilon)$ is the (unique)* initial state,
- $[n]$ *and* $[m]$ *are sets of resp.* input ports *and* output ports, *and*
- $\longrightarrow \subseteq S \times Act \times S$, *for* $Act = \{\mathsf{i}\} \times [n]\ \times \mathcal{V} \cup \{\mathsf{o}\} \times [m]\ \times \mathcal{V}$, *is a* transition relation, *given by*
  - $\big((\overline{\alpha},\overline{\beta}),x\big) \xrightarrow{\mathsf{i}\,a,v} \big((\overline{\alpha\langle a,v\rangle},\overline{\beta}),y\big)$, *if* $X([\overline{\alpha},\overline{\alpha\langle a,v\rangle}],\overline{\beta})x = y$,
  - $\big((\overline{\alpha},\overline{\beta}),x\big) \xrightarrow{\mathsf{o}\,b,v} \big((\overline{\alpha},\overline{\beta\langle b,v\rangle}),y\big)$, *if* $X(\overline{\alpha},[\overline{\beta\langle b,v\rangle},\overline{\beta}])y = x$.

Define $Seq(X) = \{\phi \in (\mathcal{I}_n \cup \mathcal{O}_m)^* \mid r \xrightarrow{\phi_0} s_1 \xrightarrow{\phi_1} s_2 \ldots \xrightarrow{\phi_{n-1}} s_n\}$, i.e the set of finite sequences of events labelling sequences of transitions of $\mathcal{A}(X)$ beginning at the initial state r.

Figure 8.5 below shows (the initial parts of) the two port automata associated to the profunctors representing the networks given in Sect. 8.1.1. All vertical arrows are output transitions and horizontal arrows are input transitions, the dotted lines indicate that the networks can receive more input tokens. Note how the two runs in process $\mathcal{A}_2$, with same input-output relation but different dependencies, are represented

We can restore the *category of elements* of the presheaf X from its associated port automaton, which thus determines X up to isomorphism [83], allowing us to work with the more

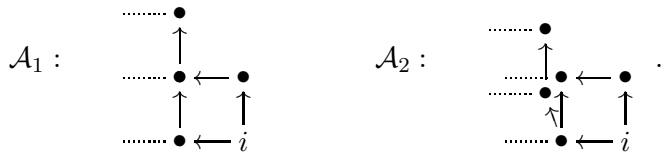


Figure 8.5: The initial parts of the two port automata associated to the profunctors representing the networks given in Sec. 8.1.1

concrete representation when convenient. Thus, we can freely confuse elements $x \in X(\overline{\alpha}, \overline{\beta})$ with their corresponding states in $\mathcal{A}(X)$.

Remarkably the axioms usually postulated for monotone port automata [103] follow for port automata of profunctors simply by functoriality.

Proposition 8.3.6 *Let* $X \colon \mathsf{H_n} \mathrel{-\!\!\!\vdash\!\!\!\to} \mathsf{H_m}$ *be a rooted port profunctor and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n], [m])$ *its associated port automaton. Then*

A1. *Receptivity:* $\forall \langle a, i, v \rangle \in \mathcal{I}_n \;\&\; s \in S.\; \exists! s' \in S.\; s \xrightarrow{\mathsf{i}\,a,v} s'$,

A2. *Monotonicity: If* $s \xrightarrow{\mathsf{o}\,b,v} t \;\&\; s \xrightarrow{\mathsf{i}\,a,v'} t'$ *then* $\exists! u \in S.\; t \xrightarrow{\mathsf{i}\,a,v'} u \;\&\; t' \xrightarrow{\mathsf{o}\,b,v} u$,

A3. *Commutativity: If* $c \neq c' \;\&\; s \xrightarrow{\mathsf{i}\,c,v} t \xrightarrow{\mathsf{i}\,c',v'} u$ *(or* $s \xrightarrow{\mathsf{o}\,c,v} t \xrightarrow{\mathsf{o}\,c',v'} u$*) then* $\exists! t' \in S.\; s \xrightarrow{\mathsf{i}\,c',v'} t' \xrightarrow{\mathsf{i}\,c,v} u$ *(or* $s \xrightarrow{\mathsf{o}\,c',v'} t' \xrightarrow{\mathsf{o}\,c,v} u$*)* .

As a consequence of the receptivity axiom, the monotonicity axiom A2 can equivalently be stated as an asymmetric commutativity axiom between input and output arrows.

Proposition 8.3.7 *Assuming axiom* A1*, axiom* A2 *of proposition 8.3.6 is equivalent to the axiom*

A2'. *If* $s \xrightarrow{\mathsf{o}\,b,v} t \;\&\; t \xrightarrow{\mathsf{i}\,a,v'} u$ *then* $\exists! t' \in S.\; s \xrightarrow{\mathsf{i}\,a,v'} t' \;\&\; t' \xrightarrow{\mathsf{o}\,b,v} u$.

Proof. *Assume axiom* A1 *and* A2*. If* $s \xrightarrow{\mathsf{o}\,b,v} t \;\&\; t \xrightarrow{\mathsf{i}\,a,v'} u$*, then by axiom* A1 *one gets that* $\exists! t' \in S.\; s \xrightarrow{\mathsf{i}\,a,v'} t'$ *and from axiom* A2 *it follows that* $\exists u' \in S.\; t' \xrightarrow{\mathsf{o}\,b,v} u'$ *and* $t \xrightarrow{\mathsf{i}\,a,v'} u'$*. Axiom* A1 *implies that* $u = u'$*. For the other direction, assume axiom* A1 *and* A2'*. If* $s \xrightarrow{\mathsf{o}\,b,v} t \;\&\; s \xrightarrow{\mathsf{i}\,a,v'} t'$*, then by axiom* A1 *one gets* $\exists u \in S.\; t \xrightarrow{\mathsf{i}\,a,v'} u$ *and from axiom* A2' *it follows that* $\exists! t'' \in S.\; s \xrightarrow{\mathsf{i}\,a,v'} t'' \;\&\; t'' \xrightarrow{\mathsf{o}\,b,v} u$*. Axiom* A1 *gives that* $t' = t''$*.* □

As a corollary, we get a mapping from port profunctors to Kahn processes.

Corollary 8.3.8 *Let* $X \colon \mathsf{H_n} \mathrel{-\!\!\!\vdash\!\!\!\to} \mathsf{H_m}$ *be a rooted port profunctor. Then* $Seq(X)$ *is a Kahn process of sort* (n, m)*.*

The above observations make rooted port profunctors look promising as a model of dataflow. However, they are a bit too general to define a causally secured trace operation. We therefor restrict attention to the class of *stable (rooted) port profunctors*. These are the profunctors for which the associated port automaton satisfies the additional axiom

A4. *Stability: If* $s \neq s', s \xrightarrow{\mathsf{i}\,a,v} t$ & $s' \xrightarrow{\mathsf{i}\,a',v'} t$ *then* $a \neq a'$ & $\exists! u.\ u \xrightarrow{\mathsf{i}\,a',v'} s$ & $u \xrightarrow{\mathsf{i}\,a,v} s'$,

Categorically, Ax. A4 is equivalent to requiring that the profunctor (when regarded as a functor $X\colon \mathsf{H_n} \to \widehat{\mathsf{H_m}}$) preserves pullbacks.

Stable rooted port profunctors define a sub symmetric monoidal category of PProf, which we will refer to as $\mathsf{PProf_s}$. We will use the notation $X\colon \mathsf{H_n} \mathrel{-\!\!\perp\!\!\!\to} \mathsf{H_m}$ to indicate that X is a profunctor in $\mathsf{PProf_s}$. Remark that for any Kahn process S there exists a stable rooted port profunctor X_S such that $Seq(X_S) = S$, i.e. the set of sequences for X_S as defined in Def. 8.3.5 equals the original process S. The profunctor X_S can be defined by simply letting $X_S(\overline{\alpha}, \overline{\beta})$ be the subset of S consisting of all sequences that restricted to only input (or output) events gives $\overline{\alpha}$ (or $\overline{\beta}$). This construction gives a functor between hom-categories $\mathsf{Kahn}[n, m]$ and $\mathsf{PProf_s}[\mathsf{H_n}, \mathsf{H_m}]$ with *Seq* as left inverse, but it does not extend to a functor between the full categories, for one thing, it does not map identities to identities.

The trace as given by the coend in (8.7) is not well defined in $\mathsf{PProf_s}$, e.g. the result of taking the trace will not always be a rooted profunctor as illustrated by the example given in the end of the previous section, which gives a presheaf with infinitely many roots. Below we will define a trace in $\mathsf{PProf_s}$ which intuitively restricts the coend to *causally secured states*. Observe that the relation $\rightsquigarrow$ defined in the explicit definition of the coend given by (8.8) can be interpreted as a relation between states connected by a chain of *internal communications* within a port automaton. More precisely, if x and x' are states of a profunctor $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \mathrel{-\!\!\!\to} \mathsf{H_m} \otimes \mathsf{H_p}$ we will let $x \Longrightarrow_p x'$ denote that $x \xrightarrow{\mathsf{o}\,c,v} \xrightarrow{\mathsf{i}\,c,v} x'$ for $c < p$, i.e. x' is reachable from x by by two transitions, the first outputs a value on a port below p and the second inputs the same value on the corresponding input port. If we now take P, Q and U in Eq. (8.8) to be respectively $\mathsf{H_n}$, $\mathsf{H_m}$ and $\mathsf{H_p}$ we get that $x \rightsquigarrow x'$ if and only if $x \Longrightarrow_p^* x'$. This leads to the following definition.

Definition 8.3.9 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \mathrel{-\!\!\perp\!\!\!\to} \mathsf{H_m} \otimes \mathsf{H_p}$ *and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n+p], [m+p])$. *Let* $N = [p \ldots n+p-1]$ *and* $M = [p \ldots m+p-1]$. *Let* $\xrightarrow{\mathsf{i}N} = \longrightarrow \cap\, S \times (\{\mathsf{i}\} \times N \times \mathcal{V}) \times S$ *and* $\xrightarrow{\mathsf{o}M} = \longrightarrow \cap\, S \times (\{\mathsf{o}\} \times M \times \mathcal{V}) \times S$. *We say that* $s \in S$ *is* p-secured *if*

$$r \xrightarrow{\mathsf{i}N}{}^* \Longrightarrow_p^* \xrightarrow{\mathsf{o}M}{}^* s \ .$$

Before giving the definition of the trace in $\mathsf{PProf_s}$, we will explore the structure of port automata obtained from port profunctors. On can easily show that they are reachable, acyclic and have at most one transition between any two states. Moreover, they satisfy two "trace-unfolding" axioms.

Lemma 8.3.10 *Let* $X\colon \mathsf{H_n} \mathrel{-\!\!\perp\!\!\!\to} \mathsf{H_m}$ *and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n]\ , [m]\ )$. *Then*

U1. *If* $t \neq t'$ & $t \xrightarrow{\mathsf{o}\,b,v} u$ & $t' \xrightarrow{\mathsf{o}\,b',v'} u$ *then* $b \neq b'$ & $\exists! s \in S.\ s \xrightarrow{\mathsf{o}\,b',v'} t$ & $s \xrightarrow{\mathsf{o}\,b,v} t'$,

U2. *If* $t \neq t'$ & $t \xrightarrow{\mathsf{i}\,a,v} u$ & $t' \xrightarrow{\mathsf{o}\,b',v'} u$ *then* $\exists! s \in S.\ s \xrightarrow{\mathsf{o}\,b',v'} t$ & $s \xrightarrow{\mathsf{i}\,a,v} t'$.

From the unfolding axioms U1 and U2 and stability A4 we get the following lemma.

Lemma 8.3.11 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \mathrel{-\!\!\perp\!\!\!\to} \mathsf{H_m} \otimes \mathsf{H_p}$ *and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n+p], [m+p])$. *Then if* $t \Longrightarrow_p u$ *and* $t' \Longrightarrow_p u$, *for* $t = \big((\overline{\alpha}, \overline{\gamma}_t, \overline{\beta}, \overline{\gamma}_t), x\big) \in S$ *and* $t' = \big((\overline{\alpha}, \overline{\gamma}_{t'}, \overline{\beta}, \overline{\gamma}_{t'}), x'\big) \in S$, *then there exists a state* $z = \big((\overline{\alpha}, \overline{\gamma}_t \wedge \overline{\gamma}_{t'}, \overline{\beta}, \overline{\gamma}_t \wedge \overline{\gamma}_{t'}), x''\big) \in S$ *such that* $z \Longrightarrow_p^* t$ *and* $z \Longrightarrow_p^* t'$.

Let $\sim_p$ denote the reflexive, symmetric and transitive closure of $\Longrightarrow_p$. The following property can be shown by induction using the lemma above.

Lemma 8.3.12 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \multimap \mathsf{H_m} \otimes \mathsf{H_p}$ *and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n+p], [m+p])$. *Then if* $t \sim_p t'$, *for* $t = \big((\overline{\alpha}, \overline{\gamma}_t, \overline{\beta}, \overline{\gamma}_t), x\big) \in S$ *and* $t' = \big((\overline{\alpha}, \overline{\gamma}_{t'}, \overline{\beta}, \overline{\gamma}_{t'}), x'\big) \in S$, *then there exists a state* $z = \big((\overline{\alpha}, \overline{\gamma}_t \wedge \overline{\gamma}_{t'}, \overline{\beta}, \overline{\gamma}_t \wedge \overline{\gamma}_{t'}), x''\big) \in S$ *such that* $z \Longrightarrow^* t$ *and* $z \Longrightarrow^* t'$.

Observation 8.3.13 Let $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \multimap \mathsf{H_m} \otimes \mathsf{H_p}$ and $\mathcal{A}(X) = (S, r, \longrightarrow, [n+p], [m+p])$. If $s \Longrightarrow_p^* t$ for $s = \big((\overline{\alpha}, \overline{\gamma}_s, \overline{\beta}, \overline{\gamma}_s), x\big)$ and $t = \big((\overline{\alpha}, \overline{\gamma}_t, \overline{\beta}, \overline{\gamma}_t), x'\big)$ then $\overline{\gamma}_s \sqsubseteq \overline{\gamma}_t$, and if $\overline{\gamma}_s = \overline{\gamma}_t$ then $s = t$.

We are now ready to show the crucial property, that for any $\sim_p$-equivalence class, there exists a minimal state from which any other state in the class is reachable by internal communication.

Lemma 8.3.14 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \multimap \mathsf{H_m} \otimes \mathsf{H_p}$ *and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n+p], [m+p])$. *Then for any* $s = \big((\overline{\alpha}, \overline{\gamma}, \overline{\beta}, \overline{\gamma}), x\big) \in S$ *there exists a unique state* $s_{\triangleright p}$ *in* $[s]_{\sim_p}$ *such that* $s_{\triangleright p} \Longrightarrow_p^* t$ *for any* $t \sim_p s$.

Proof. By Obs. 8.3.13 and Lem. 8.3.12 we get that if $s \sim_p t$ for $s = \big((\overline{\alpha}, \overline{\gamma}, \overline{\beta}, \overline{\gamma}), x\big)$ and $t = \big((\overline{\alpha}, \overline{\gamma}, \overline{\beta}, \overline{\gamma}), x'\big)$ then $s = t$. This implies that the number of elements of any $\sim_p$-equivalence class is bounded by the number of objects in $\mathsf{H_p}$, which is countable. Let $s_0, s_1, \ldots, s_i, \ldots$ be the elements of $[s]_{\sim p}$. By induction in i, we construct by using Lem. 8.3.12 elements $t_0, t_1, \ldots, t_i, \ldots$ such that $t_i \Longrightarrow_p^* s_j$ for $j \leq i$. More precisely, let $t_0 = s_0$, and assuming we have defined $t_0, \ldots, t_i$, let t_{i+1} be the element given by Lem. 8.3.12 such that $t_{i+1} \Longrightarrow_p^* t_i$ and $t_{i+1} \Longrightarrow_p^* s_{i+1}$. If $t_i = \big((\overline{\alpha}, \overline{\gamma}_i, \overline{\beta}, \overline{\gamma}_i), x\big)$ then $\overline{\gamma}_{i+1} \sqsubseteq \overline{\gamma}_j$ in $\mathsf{H_p}$. Since the category $\mathsf{H_p}$ is well founded, it follows that there exists a $k \in \omega$ s.t. $\overline{\gamma}_i = \overline{\gamma}_k$ for all $i \geq k$ and thus by Obs. 8.3.13 that $t_i = t_k$, and we can then let $s_{\triangleright p} = t_k$. For uniqueness, suppose $s_1 \neq s_2$ are both minimal states. Then $s_1 \Longrightarrow_p^* s_2$ and $s_2 \Longrightarrow_p^* s_1$, which by Obs. 8.3.13 implies that $s_1 = s_2$, □

The securedness condition has some useful equivalent formulations.

Lemma 8.3.15 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \multimap \mathsf{H_m} \otimes \mathsf{H_p}$ *and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n+p], [m+p])$. *Let* $\xrightarrow{\mathrm{i}N}$ *and* $\xrightarrow{\mathrm{o}M}$ *be defined as in Def.8.3.9. Then the following statements are equivalent*

1) $s \in S$ *is* p-secured,
2) $r \xrightarrow{\mathrm{i}N}{}^* \sim_p \xrightarrow{\mathrm{o}M}{}^* s$,
3) $r(\xrightarrow{\mathrm{i}N} \cup \sim_p \cup \xrightarrow{\mathrm{o}M})^* s$,
4) *there exists a sequence* $r \longrightarrow s_0 \longrightarrow s_1 \longrightarrow \ldots \longrightarrow s_n = s$, *such that if* $s_i = \big((\overline{\alpha}_i, \overline{\gamma}_i, \overline{\beta}_i, \overline{\delta}_i), x_i\big)$ *then* $\overline{\gamma}_i \sqsubseteq \overline{\delta}_i$ *and* $\overline{\gamma}_n = \overline{\delta}_n$.

Proof. *It follows directly that 1) implies 2), that 2) implies 3) and that 1) implies 4). That 3) implies 2) follows by repeated use of the axioms* A2' *and* A3. *To show that 2) implies 1), assume that* $r \xrightarrow{\mathrm{i}N}{}^* t \sim_p t' \xrightarrow{\mathrm{o}M}{}^* s$. *Then, by Lem. 8.3.12 there exists* z *such that* $z \Longrightarrow_p^* t$ *and* $z \Longrightarrow_p^* t'$. *Since* $t = \big((\overline{\alpha}, \epsilon, \epsilon, \epsilon), x\big)$, *it follows from Obs. 8.3.13 that* $t = z$. *That 4) implies 3) can be shown by an induction in the length of* $\overline{\gamma}_n$, *using axioms* A2' *and* A3. □

We will now finally define a trace on $\mathsf{PProf_s}$ satisfying the causal constraints of feedback.

Definition 8.3.16 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \multimap \mathsf{H_m} \otimes \mathsf{H_p}$. *Define* $\mathsf{Tr}^{\mathsf{H_p}}_{\mathsf{H_n},\mathsf{H_m}}(X)\colon \mathsf{H_n} \multimap \mathsf{H_m}$, *the trace of* X *to be given by*

$$\mathsf{Tr}^{\mathsf{H_p}}_{\mathsf{H_n},\mathsf{H_m}}(X)(\overline{\alpha},\overline{\beta}) \cong \biguplus_{\overline{\gamma}\in\mathsf{H_p}} \{\, x \in X(\overline{\alpha},\overline{\gamma},\overline{\beta},\overline{\gamma}) \mid x \text{ is } p\text{-secured}\}_{/\sim_p}, \tag{8.11}$$

where the action on arrows is defined as for the coend. We will often abbreviate $\mathsf{Tr}^{\mathsf{H_p}}_{\mathsf{H_n},\mathsf{H_m}}$ *to* $\mathsf{Tr}^p_{n,m}$ *or just* Tr^p.

For the above definition to be well defined we must first show that if $h\colon \overline{\gamma} \to \overline{\gamma}'$ and $j\colon \overline{\delta}' \to \overline{\delta}$ are arrows of respectively $\mathsf{H_n}$ and $\mathsf{H_m}$ and $x \in X\big((\overline{\gamma},\overline{\alpha}),(\overline{\alpha},\overline{\delta})\big)$ is p-secured, then $y = X\big((h,1_{\overline{\alpha}}),(1_{\overline{\alpha}},j)\big)x$ is p-secured as well. Assume x is p-secured and let $z = X\big((h,1_{\overline{\alpha}}),(1_{\overline{\alpha}},1_{\overline{\delta}})\big)x$, so $y = X\big((1_{\overline{\gamma}'},1_{\overline{\alpha}}),(1_{\overline{\alpha}},j)\big)z$. Then $x \xrightarrow{\mathsf{i}\phi_1}\xrightarrow{\mathsf{i}\phi_2} \cdots \xrightarrow{\mathsf{i}\phi_n} z$ for $\overline{\gamma}' = \overline{\gamma\phi_1\phi_2\ldots\phi_n}$ and $y \xrightarrow{\mathsf{o}\psi_1}\xrightarrow{\mathsf{o}\psi_2} \cdots \xrightarrow{\mathsf{o}\psi_n} z$ for $\overline{\delta} = \overline{\delta'\psi_1\psi_2\ldots\psi_n}$. By the assumption that x is p-secured it follows directly from 3) of Lem. 8.3.15 that z is p-secured. Using 2) of Lem. 8.3.15 and Ax. $\mathsf{U1}$ repeatedly one can then easily show that y is p-secured as well. Finally we must show that if $\tau\colon X \to Y$ is a natural transformation in $\mathsf{PProf_s}[\mathsf{H_n}\otimes\mathsf{H_p},\mathsf{H_m}\otimes\mathsf{H_p}]$ and $x \in X\big((\overline{\gamma},\overline{\alpha}),(\overline{\alpha},\overline{\delta})\big)$ is p-secured, then $y = \tau_{(\overline{\gamma},\overline{\alpha}),(\overline{\alpha},\overline{\delta})}x \in Y\big((\overline{\gamma},\overline{\alpha}),(\overline{\alpha},\overline{\delta})\big)$ is p-secured as well. If

$$r_X \xrightarrow{\mathsf{i}N} {}^{n_1} \Longrightarrow^{n_2}_p \xrightarrow{\mathsf{o}M} {}^{n_3}x \ ,$$

one can show that

$$r_Y \xrightarrow{\mathsf{i}N} {}^{n_1} \Longrightarrow^{n_2}_p \xrightarrow{\mathsf{o}M} {}^{n_3}y \ ,$$

by a simple induction in $n_1+n_2+n_3$ using naturality of τ and the property of the stability axiom $\mathsf{A4}$ that if $s \neq s', s \xrightarrow{\mathsf{i}\,a,v} t$ & $s' \xrightarrow{\mathsf{i}\,a',v'} t$ then $a \neq a'$.

The trace can be expressed on port automata as follows.

Proposition 8.3.17 *Let* $X\colon \mathsf{H_n}\otimes\mathsf{H_p} \multimap \mathsf{H_m}\otimes\mathsf{H_p}$ *and* $\mathcal{A}(X) \cong (S, r, \longrightarrow, [n+p], [m+p])$. *Then* $\mathcal{A}(\mathsf{Tr}^p_{n,m}(X)) = (S_{\sim p}, [r]_{\sim p}, \longrightarrow_{\sim p}, [n]\ , [m]\ )$, $S_{\sim p} = \{s \in S \mid s \text{ is } p\text{-secured}\}_{/\sim}$ *and*

$$[s]_{\sim p} \xrightarrow{\mathsf{i}\,a,v}_{\sim p} [s']_{\sim p} \text{ if } s \xrightarrow{\mathsf{i}\,a+p,v} s', \text{ for } a \in [n] \text{ and}$$

$$[s]_{\sim p} \xrightarrow{\mathsf{o}\,b,v}_{\sim p} [s']_{\sim p} \text{ if } s \xrightarrow{\mathsf{o}\,b+p,v'} s', \text{ for } b \in [m].$$

By quotienting the state space by the equivalence relation $\sim_p$, a forward computation in the traced automaton may originate from a computation in which internal transitions can be made backwards. However, using Lem. 8.3.14 we can replace the equivalence classes with the minimal representatives given in the lemma, obtaining the formulation below where any forward computation in the traced automaton originates from a forward computation in the underlying automaton.

Lemma 8.3.18 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \multimap \mathsf{H_m} \otimes \mathsf{H_p}$ *and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n+p], [m+p])$.

$$s \sim_p \xrightarrow{\mathsf{i}\, a+p,v} \sim_p s', \text{ for } a \in [n] \text{ if and only if } s_{\triangleright p} \xrightarrow{\mathsf{i}\, a+p,v} s'_{\triangleright p}, \text{ for } a \in [n] \text{ and} \tag{8.12}$$

$$s \sim_p \xrightarrow{\mathsf{o}\, b+p,v} \sim_p s', \text{ for } b \in [m] \text{ if and only if } s_{\triangleright p} \Longrightarrow_p^* \xrightarrow{\mathsf{i}\, b+p,v} s'_{\triangleright p}, \text{ for } b \in [m]. \tag{8.13}$$

Proof. The if direction of both (8.12) and (8.13) is immediate. For the only if direction of (8.12) we first show that if $s_{\triangleright p} \xrightarrow{\mathsf{i}\, a+p} t$ then $t = t_{\triangleright p}$ by contradiction. Suppose that there exists t' such that $t' \Longrightarrow_p t$. Then by axioms $\mathsf{U1}$, $\mathsf{U2}$ it follows that there exists $s' \Longrightarrow_p s_{\triangleright p}$. But this contradicts the minimality of $s_{\triangleright p}$, so we conclude that there exists no t' such that $t' \Longrightarrow_p t$,i.e. $t = t_{\triangleright p}$. □

Proposition 8.3.19 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \multimap \mathsf{H_m} \otimes \mathsf{H_p}$ *and* $\mathcal{A}(X) = (S, r, \longrightarrow, [n+p], [m+p])$. *Then* $\mathcal{A}(\mathsf{Tr}^p_{n,m}(X)) \cong (S_p\, r, \longrightarrow_p, [m]\ , [n]\ )$, $S_p = \{s_{\triangleright p} \in S \mid s \text{ is } p\text{-secured}\}$, *where* $s_{\triangleright p}$ *is given as in Lem. 8.3.14 and*

$$s_{\triangleright p} \xrightarrow{\mathsf{i}\, a,v}_p s'_{\triangleright p} \text{ if } s_{\triangleright p} \xrightarrow{\mathsf{i}\, a+p,v} s'_{\triangleright p}, \text{ for } a \in [n] \text{ and}$$

$$s_{\triangleright p} \xrightarrow{\mathsf{o}\, b,v}_p s'_{\triangleright p} \text{ if } s_{\triangleright p} \Longrightarrow_p^* \xrightarrow{\mathsf{o}\, b+p,v'} s'_{\triangleright p}, \text{ for } b \in [m].$$

Proof. *Follows from Prop. 8.3.17 and Lem. 8.3.18.*

□

Intuitively this expresses that internal computation cannot be observed in itself.

We are now ready to state the main theorem, that the trace operator given in Def. 8.3.16 satisfy the axioms of a traced monoidal category up to isomorphism. The proof can be found in App. 8.6

Theorem 8.3.20 *With the trace operator given in Def. 8.3.16,* $\mathsf{PProf_s}$ *satisfies up to isomorphism the axioms of a traced monoidal category.*

Proposition 8.3.21 *The map Seq given in Def. 8.3.5 defines the action on arrows of a functor* $Seq\colon \mathsf{PProf_s} \to \mathsf{Kahn}$, *that preserves the traced monoidal structure, on objects mapping* $\mathsf{H_n}$ *to* n. Proof. *(Sketch) To show that trace is preserved, we use Lem. 8.3.15 formulation 4) of securedness and Prop. 8.3.19.* □

We end this section with an important remark, namely that the secured trace can be defined as the composition of two functors on hom-categories; first a functor restricting to secured states and then a *colimit*, by using a standard construction of the *subdivision category* [82] which allows any coend to be expressed as a colimit. The details of this definition of the trace operator can be found in App. 8.7. Below we will benefit from the colimit formulation of the trace.

8.4 Some Consequences

We will briefly go through some of the consequences of the categorical semantics of dataflow given in the two previous sections.

8.4.1 A Bisimulation Congruence

The presentation of models for concurrency as categories allows us to apply a general notion of bisimulation from spans of open maps proposed in [71]. The general idea is to identify a *path category* $\mathsf{P} \hookrightarrow \mathsf{M}$ as a subcategory of the model M, with objects representing runs or histories and morphisms compatible extensions of these. For a presheaf model $\hat{\mathsf{P}}$ the canonical choice of path category is the category P under the Yoneda embedding. A morphism is then said to be *open* if it reflects extensions of histories, and two objects are said to be *open map bisimilar* if they are connected by a span of (surjective) open maps.

Definition 8.4.1 *Let $f\colon X \to Y$ be a morphism in $\hat{\mathsf{P}}$. Then f is* open *if for any arrow $e\colon P \to Q$ of P the square*

$$\begin{array}{ccc} X(Q) & \xrightarrow{Xe} & X(P) \\ {\scriptstyle f_Q}\downarrow & & \downarrow{\scriptstyle f_P} \\ Y(Q) & \xrightarrow{Ye} & Y(P) \end{array}$$

is a quasi-pullback. Two presheaves X and Y in $\hat{\mathsf{P}}$ are open map bisimilar *if they are related by a span of surjective open maps.*

Recall that a port profunctor $X\colon \mathsf{H_n} \mathrel{-\!\!\perp\!\!\to} \mathsf{H_m}$ can be viewed as a presheaf in $\widehat{\mathsf{H_n}^{\mathsf{op}} \times \mathsf{H_m}}$ and so we get a canonical notion of bisimulation from open maps as defined above.

As for the presheaves as transition systems in [147], the open map bisimulation can be characterised as a back-&-forth bisimulation between the associated port automata.

Proposition 8.4.2 *Let $X_i\colon \mathsf{H_n} \mathrel{-\!\!\perp\!\!\to} \mathsf{H_m}$ and $\mathcal{A}(X_i) = (S_i, r_i, \longrightarrow_i, [n]\ , [m]\ )$ for $i \in \{1,2\}$. X_1 and X_2 are open map bisimilar iff $\mathcal{A}(X_1)$, $\mathcal{A}(X_2)$ are* back-&-forth bisimilar*: There exists a relation $R \subseteq S_1 \times S_2$ such that $(r_1, r_2) \in R$ and*

- $(s,s') \in R \;\&\; t \xrightarrow{\phi}_1 s \Rightarrow \exists t'.\ t' \xrightarrow{\phi}_2 s' \;\&\; (t,t') \in R$,
- $(s,s') \in R \;\&\; s \xrightarrow{\phi}_1 t \Rightarrow \exists t'.\ s' \xrightarrow{\phi}_2 t' \;\&\; (t,t') \in R$,
- $(s,s') \in R \;\&\; t' \xrightarrow{\phi}_2 s' \Rightarrow \exists t.\ t \xrightarrow{\phi}_1 s \;\&\; (t,t') \in R$,
- $(s,s') \in R \;\&\; s' \xrightarrow{\phi}_2 t' \Rightarrow \exists t.\ s \xrightarrow{\phi}_1 t \;\&\; (t,t') \in R$.

It is important to check that our notion of bisimulation on $\mathsf{PProf_s}$ is a congruence with respect to the operations tensor and trace. Here we can exploit some general properties of open maps and so bisimulation on presheaves: the product of (surjective) open maps in a presheaf category is (surjective) open [69]; any colimit-preserving functor between presheaf categories preserves (surjective) open maps [23]. The proof that trace on $\mathsf{PProf_s}$ preserves

Figure 8.6: A process with bi-directional I/O implemented by an uni-directional process. Dotted lines indicate channels that play the opposite role in the higher-order model

bisimulation uses the latter property, exploiting the fact that trace can be expressed as a colimit, first showing that $\mathcal{S}$ as a functor between presheaf categories preserves open maps. The proof of the corresponding result for tensor rests on a construction of tensor from more basic functors, which are all colimit-preserving and so preserve (surjective) open maps. Using the simple fact that bisimulation is a congruence with respect to sequential composition with a symmetry together with the generalised yanking property we can conclude that bisimulation is a congruence with respect to all of the operations on networks as stated in the theorem below. The details of the proof can be found in App. 8.7.

Theorem 8.4.3 *Open map bisimulation in* $\mathsf{PProf_s}$ *is a congruence with respect to sequential composition, tensor and trace.*

By placing dataflow within profunctors and the broader class of presheaf models, constructions of dataflow could be mixed with constructions from other paradigms of concurrent computation such as those traditionally from CCS-like process calculi. As an example, a kind of synchronous communication can be represented by the product of presheaves. In this richer world of constructions bisimulation would appear to be the more suitable equivalence.

8.4.2 Higher-order Dataflow via Geometry of Interaction

The Geometry of interaction program was invented by Girard in his analysis of the fine structure of cut elimination [44, 45]. His basic insight was that higher order structure could be understood in terms of trace but this understanding was hidden in the mathematical setting - Hilbert spaces and traces of operators - that he used. In [70] Joyal, Street, and Verity and independently Abramsky [3] (see also [6]) gave the categorical expression of the idea which was that a traced monoidal category could be "completed" to yield a compact closed category. As such it gives a method for realizing higher-order constructs in terms of feedback. In our setting one takes the categories Kahn and $\mathsf{PProf_s}$ and constructs compact-closed categories HKahn and $\mathsf{HPProf_s}$ which then serve as the interpretations of higher-order Kahn processes and port profunctors.

We will just give the main definition, for more details see [70, 3]. Essentially, one obtains a higher-order model by working with processes with bi-directional "input" and "output". These processes are implemented by uni-directional processes of the underlying category in the obvious way, regarding negative, i.e. reversed, input channels as output channels and negative output as input.

Definition 8.4.4 *Given a traced monoidal category* $\mathcal{C}$ *we define a new category* $\mathcal{G}(\mathcal{C})$ *as follows. The objects of* $\mathcal{G}(\mathcal{C})$ *are pairs of objects* (A^+, A^-) *of* $\mathcal{C}$*. A morphism* $f : (A^+, A^-) \to (B^+, B^-)$ *of* $\mathcal{G}(\mathcal{C})$ *is a* $\mathcal{C}$*-morphism* $f : A^+ \otimes B^- \to B^+ \otimes A^-$*, as illustrated in Fig. 8.6. Composition is implemented using composition, trace and symmetries of* $\mathcal{C}$ *to connect channels*

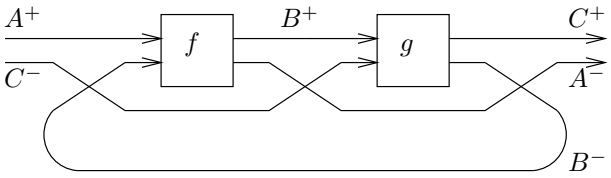


Figure 8.7: Implementation of composition in the higher-order model using symmetries and trace

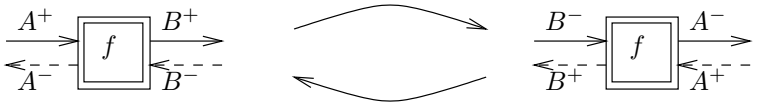


Figure 8.8: The duality is obtained by swapping the roles of the channels

with same polarity, ie. for $g : (B^+, B^-) \to (C^+, C^-)$*,* $f; g$ *is implemented, as illustrated in Fig. 8.7, by*

$$\mathsf{Tr}^{B^-}_{A^+\otimes C^-, C^+\otimes A^-}((I_{A^+} \otimes \sigma); (f \otimes I_{C^-}); (I_{B^+} \otimes \sigma'); (g \otimes I_{A^-}); (I_{C^+} \otimes \sigma'')) \ ,$$

for the appropriate symmetries σ*,* σ' *and* σ''*.*

Note that $\mathcal{C}$ embeds into $\mathcal{G}(\mathcal{C})$ as arrows with no negative flow, mapping objects A to (A, I). A symmetric monoidal structure $\odot$ is defined on objects by $(A^+, A^-) \odot (B^+, B^-) = (A^+ \otimes B^+, B^- \otimes A^-)$. An obvious duality is defined on objects by $(A^+, A^-)^* = (A^-, A^+)$, and on arrows by swapping the roles of channels as illustrated in Fig. 8.8.

This defines a contravariant functor $(-)^* : \mathcal{G}(\mathcal{C}) \to \mathcal{G}(\mathcal{C})$. Internal hom sets are given by $(A^+, A^-) \multimap (B^+, B^-) = (B^+, B^-) \odot (A^+, A^-)^*$, giving an involution as illustrated by Fig. 8.9.

We can now directly apply the above construction to the category Kahn, obtaining a category $\mathcal{G}(\mathsf{Kahn})$ of *higher order Kahn processes*, which we will denote by HKahn. Since $\mathsf{PProf}_\mathsf{s}$ is a bicategory, only satisfying the axioms of a TMC up to isomorphism, $\mathcal{G}(\mathsf{PProf}_\mathsf{s})$ will not be a category, e.g. composition is only associative up to isomorphism. There may be several ways around this problem. It is likely to be the case, that by making precise what it means to be a traced monoidal bicategory one can show that $\mathcal{G}(\mathsf{PProf}_\mathsf{s})$ is a compact closed bicategory. This should be related to the work in [76]. Another possibility is to consider the quotient of $\mathsf{PProf}_\mathsf{s}$ with respect to open map bisimulation, analogous to the definition of the category $\mathcal{ASP}roc$ in [4, 43]. That is, instead of $\mathsf{PProf}_\mathsf{s}$ use the category with objects being path categories as usual, but taking arrows to be equivalence classes with respect to open

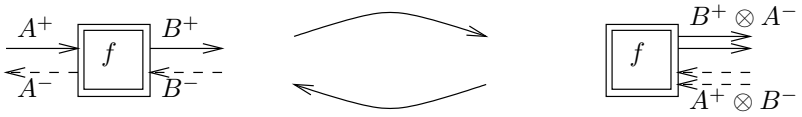


Figure 8.9: Involution

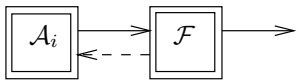


Figure 8.10: The fork process $\mathcal{F}$ regarded as a higher-order process, applied to the automata $\mathcal{A}_i$

map bisimulation. By Thm. 8.4.3 this is indeed well defined and it is easy to check that we get a traced (strict) symmetric monoidal category.

Below we will simply let $\mathsf{HPProf_s}$ refer to a higher order (possibly bi)category constructed using the $\mathcal{G}$ construction without making explicit how the problem is managed.

Since the functor *Seq* from $\mathsf{PProf_s}$ to Kahn preserves tensor and trace it extends to one between the higher-order categories.

Proposition 8.4.5 *We have a functor* $HSeq\colon \mathsf{HPProf_s} \to \mathsf{HKahn}$*, defined using Seq on the base category.*

The higher order structure of HKahn and $\mathsf{HPProf_s}$ has a very intuitive interpretation in the underlying categories Kahn and $\mathsf{PProf_s}$ as plugging networks into contexts. As an example, the fork process $\mathcal{F}\colon \mathsf{H_1} \mathrel{-\!\!\perp\!\!\to} \mathsf{H_1} \otimes \mathsf{H_1}$ which was used in definining the context $\mathcal{C}[-]$ of Sect. 8.1.1 implements the higher order process $\mathcal{F}\colon (\mathsf{H_1}, \mathsf{H_1}) \to (\mathsf{H_1}, I)$ which can be regarded as the process $\mathcal{F}\colon (\mathsf{H_1} \multimap \mathsf{H_1}) \to \mathsf{H_1}$ writing $\mathsf{H_1}$ as short for $(\mathsf{H_1}, I)$. The processes $\mathcal{A}_i\colon \mathsf{H_1} \mathrel{-\!\!\perp\!\!\to} \mathsf{H_1}$ implements higher order processes $\mathcal{A}_i\colon (\mathsf{H_1}, I) \to (\mathsf{H_1}, I)$ which by the involution can be regarded as processes $\mathcal{A}_i\colon I \to (\mathsf{H_1} \multimap \mathsf{H_1})$ (again writing $\mathsf{H_1}$ as short for $(\mathsf{H_1}, I)$). Now the processes $\mathcal{C}[\mathcal{A}_i]$ are simply the processes $\mathcal{A}_i; \mathcal{F}$ obtained by composition as illustrated in Fig. 8.10.

8.5 Concluding Remarks

The upshot of the work in this paper is a treatment of dataflow that unifies different phenomena - asynchrony and synchrony in our case - and different viewpoints of dataflow networks: dataflow composition as relational composition, dataflow processes as categorical constructs and the concrete views of dataflow networks as port automata and as sequences of events encoding causality. In particular, dataflow feedback is shown as an instance of a trace operation in a category and this allows one to adapt the ideas from the geometry of interaction program to give a smooth treatment of higher-order processes. The higher-order models should be compared to the work in [4]. It also remains to explore systematically the full family of models for dataflow, relating automata, event structure and traces-based models to the relational model, following the pattern set in [146]. Work is underway on a bicategory of (higher order) port automata. This will provide further operational support to the trace on port profunctors and help in the understanding of independence at higher-order. Early attempts have been made to incorporate fairness into the profunctor model; it is hoped to exploit independence along the lines in [26] and include maximal or *completed* observations.

8.6 Traced Monoidal Properties of $\mathsf{PProf_s}$

We begin with a lemma stating the simple fact that symmetries correspond simply to reordering of channels when viewed as automata. Recall that we refer to symmetries by $\sigma_{n,n'} \colon \mathsf{H_n} \otimes \mathsf{H_{n'}} \to \mathsf{H_{n'}} \otimes \mathsf{H_n}$. We will abuse notation and also use $\sigma_{n,n'} \colon [n+n'] \to [n'+n]$ to refer to the corresponding permutation on $[n+n']$, mapping $j \in [n+n']$ to $j+n'$ if $j < n$ and to $j-n$ if $j \geq n$.

Lemma 8.6.1 *Let $X \colon \mathsf{H_n} \otimes \mathsf{H_{n'}} \to \mathsf{H_m} \otimes \mathsf{H_{m'}}$ and $\mathcal{A}(X) = (S, r, \longrightarrow, [n+n'], [m+m'])$. Then $\mathcal{A}(\sigma_{n',n}; X; \sigma_{m,m'}) \cong (S, r, \longrightarrow_\sigma, [n'+n], [m'+m])$, where the transition relation $\longrightarrow_\sigma$ is defined by*

- $s \xrightarrow{\mathsf{i}\,av}_\sigma s'$ *if* $s \xrightarrow{\mathsf{i}\,a'v} s'$, *for* $a' = \sigma_{n',n}(a)$ *and*
- $s \xrightarrow{\mathsf{o}\,bv}_\sigma s'$ *if* $s \xrightarrow{\mathsf{o}\,b'v} s'$, *for* $b = \sigma_{m,m'}(b')$.

We now show that the generalised yanking condition is satisfied.

Lemma 8.6.2 *Let $X \colon \mathsf{H_n} \to \mathsf{H_p}$ and $Y \colon \mathsf{H_p} \to \mathsf{H_m}$. Then there is a (natural) isomorphism*

$$\mathsf{Tr}^p((X \otimes Y); \sigma_{p,m}) \cong X; Y \ .$$

Proof. *Using Lem. 8.6.1 it is easy to show that for all $s = \big((\overline{\alpha}, \overline{\gamma}, \overline{\beta}, \overline{\gamma}), x\big)$ of $\mathcal{A}\big((X \otimes Y); \sigma_{p,m}\big)$, s is p-secured. This gives that $\mathsf{Tr}^p((X \otimes Y); \sigma_{p,m}) = Tr^{\mathsf{H_p}}_{\mathsf{H_n},\mathsf{H_m}}((X \otimes Y); \sigma_{p,m})$. The desired result then follows from Prop. 8.3.4.* □

This is the first step in showing that the trace as given by Def. 8.3.16 satisfies all the properties of a traced monoidal category. Next we show that trace distributes through tensor.

Lemma 8.6.3 *Let $X \colon \mathsf{H_n} \otimes \mathsf{H_p} \dashv\!\perp\!\mapsto \mathsf{H_m} \otimes \mathsf{H_p}$ and $Y \colon \mathsf{H_{n'}} \to \mathsf{H_{m'}}$. Then there is a (natural) isomorphism*

$$\mathsf{Tr}^p_{m'+m,n'+n}(Y \otimes X) \cong Y \otimes \mathsf{Tr}^p_{m,n}(X) \ .$$

Proof. *Let $N = [p \ldots n+p-1]$, $M = [p \ldots m+p-1]$, $N' = [n+p \ldots n'+n+p-1]$, $M' = [m+p \ldots m'+n+p-1]$ and define $\xrightarrow{\mathsf{i}N}$, $\xrightarrow{\mathsf{i}N'}$, $\xrightarrow{\mathsf{o}M}$ and $\xrightarrow{\mathsf{o}M'}$ as in Def. 8.3.9. We then show that for all states s of $\mathcal{A}\big(Y \otimes X\big)$, it holds that*

$$r \xrightarrow{\mathsf{i}N \cup N'}{}^* t \Longrightarrow_p{}^* t' \xrightarrow{\mathsf{o}M \cup M'}{}^* s$$

if and only if

$$r \xrightarrow{\mathsf{i}N}{}^* t \Longrightarrow_p{}^* t' \xrightarrow{\mathsf{o}M}{}^* \xrightarrow{\mathsf{i}N'}{}^* \xrightarrow{\mathsf{o}M'}{}^* s \ .$$

It follows that the (natural) isomorphism

$$Tr^p_{m'+m,n'+n}(Y \otimes X) \cong Y \otimes Tr^p_{m,n}(X) \ ,$$

which holds for the trace as given by the coend restricts to one for the restricted coend. □

The proof of the Bekic property follows the same procedure, showing that the (natural) isomorphism existing in PProf expressing the Bekic property[5] restricts to a (natural) isomorphism in $\mathsf{PProf_s}$. Here we make crucial use of the stability condition. Indeed there exist a simple, non-stable port profunctor for which the Bekic property is not satisfied.

Lemma 8.6.4 *Let* $X : \mathsf{H_n} \otimes \mathsf{H_p} \otimes \mathsf{H_q} \multimap \mathsf{H_m} \otimes \mathsf{H_p} \otimes \mathsf{H_q}$. *Let* x *be a state of* $\mathcal{A}(X)$. *Then*

$$x \text{ is (p+q)-secured } \Leftrightarrow [x]_{\sim_q} \text{ is a p-secured state in } \mathsf{Tr}^q_{n+p,m+p}(X).$$

Proof. Let $N = [p+q \ldots n+p+q-1]$, $M = [p+q \ldots m+p+q-1]$ and $P = [q \ldots p+q-1]$. Let $\Longrightarrow_{p_{\sim q}} =_{def} (\sim_q \xrightarrow{\mathsf{o}\,c,v} \sim_q \xrightarrow{\mathsf{i}\,c,v} \sim_q) \cup \sim_q$, for $c \in P$. This can be read as "feedback of (atmost) one value on a port in P upto back&forth communication on the ports below q". Let $\xrightarrow{\mathsf{i}N}_{\sim q} =_{def} \sim_q \xrightarrow{\mathsf{i}\,a,v} \sim_q$ for $a \in N$ and $\xrightarrow{\mathsf{o}M}_{\sim q} =_{def} \sim_q \xrightarrow{\mathsf{o}\,b,v} \sim_q$ for $b \in M$. Let $\xrightarrow{\mathsf{i}N}$ and $\xrightarrow{\mathsf{o}M}$ be defined as in Def. 8.3.9. From Lem. 8.3.15 it is not difficult to get that $[x]_{\sim_q}$ is p-secured in $\mathsf{Tr}^q_{n+p,m+p}(X)$ if and only if

$$r(\xrightarrow{\mathsf{i}N\cup P} \cup \sim_q \cup \xrightarrow{\mathsf{o}M\cup P})^* x \quad \text{and} \quad r(\xrightarrow{\mathsf{i}N}_{\sim q} \cup \sim_{p\sim q} \cup \xrightarrow{\mathsf{o}M}_{\sim q})^* x \tag{8.14}$$

where $\sim_{p\sim q}$ denote the reflexive, symmetric and transitive closure of $\Longrightarrow_{p_{\sim q}}$. The first part simply says that $[x]_{\sim_q}$ is a state of $\mathsf{Tr}^q_{n+p,m+p}(X)$. From Lem. 8.3.18 it follows that $\sim_q \xrightarrow{\mathsf{o}\,c,v} \sim_q \xrightarrow{\mathsf{i}\,c,v} \sim_q = \sim_q \xrightarrow{\mathsf{o}\,c,v} \xrightarrow{\mathsf{i}\,c,v} \sim_q$, for $c \in P$. Since by definition $\sim_q \subseteq \sim_{p+q}$ we get $\sim_{p\sim q} = \sim_{p+q}$. This implies that (8.14) is equivalent to

$$r(\xrightarrow{\mathsf{i}N\cup P} \cup \sim_q \cup \xrightarrow{\mathsf{o}M\cup P})^* x, \quad \text{and} \quad r(\xrightarrow{\mathsf{i}N} \cup \sim_{p+q} \cup \xrightarrow{\mathsf{o}M})^* x \tag{8.15}$$

which by definition is equivalent to

$$r(\xrightarrow{\mathsf{i}N} \cup \sim_{p+q} \cup \xrightarrow{\mathsf{o}M})^* x, \tag{8.16}$$

which by Def. 8.3.9 is equivalent to saying that x is (p+q)-secured. □ The lemma above gives us that the Bekic property holds for the restricted coend in $\mathsf{PProf_s}$ as stated below.

Proposition 8.6.5 *Let* $X : \mathsf{H_n} \otimes \mathsf{H_p} \otimes \mathsf{H_q} \multimap \mathsf{H_m} \otimes \mathsf{H_p} \otimes \mathsf{H_q}$. *Then there is a (natural) isomorphism*

$$\mathsf{Tr}^q_{n+p,m+p}(X) \cong \mathsf{Tr}^p_{n,m}(\mathsf{Tr}^q_{n+p,m+p}(X))$$

Proof. It is a fact [82] that there is a (natural) isomorphism

$$Tr^q_{n+p,m+p}(X) \cong Tr^p_{n,m}(Tr^q_{n+p,m+p}(X))$$

in PProf. By Lem. 8.6.4 it follows that this isomorphism restricts to the desired isomorphism in $\mathsf{PProf_s}$. □ Using the results above and those of Sec. 8.3.2 we can now show that the axioms of a traced monoidal category as given in Def. 8.2.1 are satisfied by PProf and the trace operator given in Def. 8.3.16 up to isomorphism. The yanking axiom follows directly from Lem. 8.6.2 taking both X and Y to be identities. Superposing follows directly from Lem. 8.6.3 taking Y to be an identity. The Bekic (I) axiom is proven in Prop. 8.6.5 and the Bekic (II) axiom is easily shown to hold. The naturality axioms can be shown following the same pattern as the other axioms, restricting the corresponding (natural) isomorphisms that are known to exist in PProf.

[5] That a "double integral" can be obtained as an "iterated" integral (dual to the proposition p.226 [82])

8.7 Colimit formulation of Trace

We give here the formal colimit definition of the trace operator in $\mathsf{PProf_s}$ mentioned in the end of Sec. 8.3.2. The intuition is that the trace can be defined as the composition of two functors on presheaf categories, the first restricting to secured states and the the second, a colimit, hiding the internal communication.

We begin with the standard construction of the *subdivision category* [82] for a category $\mathsf{H_p}$. Note the definition here is the dual to that in [82] since we are concerned with coends and not ends. For a category $\mathsf{H_p}$, the subdivision category ${\mathsf{H_p}}^{\S}$ has as objects all arrows $f\colon \overline{\alpha} \to \overline{\beta}$ of $\mathsf{H_p}$ (i.e. $f = [\overline{\alpha}, \overline{\beta}]$.) For each such object f, it has two arrows $f_{\mathsf{o}}\colon f \to 1_\alpha$ and $f_{\mathsf{i}}\colon f \to 1_\beta$, i.e. $[\overline{\alpha}, \overline{\alpha}] \xleftarrow{f_{\mathsf{o}}} [\overline{\alpha}, \overline{\beta}] \xrightarrow{f_{\mathsf{i}}} [\overline{\beta}, \overline{\beta}]$. These are the only non-identity arrows. Now we define a functor $\mathcal{S}\colon \mathsf{PProf_s}[\mathsf{H_n} \otimes \mathsf{H_p}, \mathsf{H_m} \otimes \mathsf{H_p}] \to \mathsf{Prof}[\mathsf{H_n} \times {\mathsf{H_p}}^{\S}, \mathsf{H_m}]$ as in the standard construction, except we restrict to secured states. Let X be a profuncor in $\mathsf{PProf_s}[\mathsf{H_n} \otimes \mathsf{H_p}, \mathsf{H_m} \otimes \mathsf{H_p}]$, i.e. $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \times (\mathsf{H_m} \otimes \mathsf{H_p})^{\mathsf{op}} \to \mathsf{Set}$. Define a functor $\mathcal{S}(X)\colon \mathsf{H_n} \times {\mathsf{H_p}}^{\S} \times {\mathsf{H_m}}^{\mathsf{op}} \to \mathsf{Set}$ as follows. For $(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})$ an object in $\mathsf{H_n} \times {\mathsf{H_p}}^{\S} \times {\mathsf{H_m}}^{\mathsf{op}}$ define

$$\mathcal{S}(X)(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta}) = \{x \in X\big((\overline{\gamma}, \overline{\alpha}), (\overline{\beta}, \overline{\delta})\big) \mid X\big((1_{\overline{\gamma}}, 1_{\overline{\alpha}}), ([\overline{\beta}, \overline{\alpha}], 1_{\overline{\delta}})\big)x \text{ is } p\text{-secured } \},$$

where we have implicitly used the equivalence between $\mathsf{Prof}[\mathsf{H_{n+p}}, \mathsf{H_{m+p}}]$ and $\mathsf{Prof}[\mathsf{H_n} \times \mathsf{H_p}, \mathsf{H_m} \times \mathsf{H_p}]$.

For $h\colon \overline{\gamma} \to \overline{\gamma}'$ and $j\colon \overline{\delta}' \to \overline{\delta}$ arrows of respectively $\mathsf{H_n}$ and $\mathsf{H_m}$, define for $x \in \mathcal{S}(X)(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})$

$$\mathcal{S}(X)(h, [\overline{\alpha}, \overline{\beta}]_{\mathsf{o}}, j)x = X\big((h, 1_{\overline{\alpha}}), ([\overline{\beta}, \overline{\alpha}], j)\big)x \quad \text{and}$$

$$\mathcal{S}(X)(h, [\overline{\alpha}, \overline{\beta}]_{\mathsf{i}}, j)x = X\big((h, [\overline{\alpha}, \overline{\beta}]), (1_{\overline{\beta}}, j)\big)x.$$

For $\tau\colon X \to Y$ is a natural transformation in $\mathsf{PProf_s}[\mathsf{H_n} \otimes \mathsf{H_p}, \mathsf{H_m} \otimes \mathsf{H_p}]$ define $\mathcal{S}(\tau)\colon \mathcal{S}(X) \to \mathcal{S}(Y)$ by

$$\mathcal{S}(\tau)_{(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})}x = \tau_{(\overline{\gamma}, \overline{\alpha}, \overline{\beta}, \overline{\delta})}x,$$

for $(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})$ an object in $\mathsf{H_n} \times {\mathsf{H_p}}^{\S} \times {\mathsf{H_m}}^{\mathsf{op}}$.

To check this is well defined we proceed as for Def. 8.3.16 and in addition use the fact that if $x \in X\big((\overline{\gamma}, \overline{\alpha}), (\overline{\beta}, \overline{\delta})\big)$ and $X\big((1_{\overline{\gamma}}, 1_{\overline{\alpha}}), ([\overline{\beta}, \overline{\alpha}], 1_{\overline{\delta}})\big)x$ is p-secured, then $X\big((1_{\overline{\gamma}}, [\overline{\alpha}, \overline{\beta}]), (1_{\overline{\beta}}, 1_{\overline{\delta}})\big)x$ is p-secured.

Proposition 8.7.1 *Let* $X\colon \mathsf{H_n} \otimes \mathsf{H_p} \nrightarrow \mathsf{H_m} \otimes \mathsf{H_p}$. *Then*

$$\mathsf{Tr}^p_{n,m}(X) \cong \mathrm{Colim}_{{\mathsf{H_p}}^{\S}} \mathcal{S}(X), \tag{8.17}$$

where ${\mathsf{H_p}}^{\S}$ *is the* subdivision category *of* $\mathsf{H_p}$ *as defined above.*

Proof. As for the standard construction. □

8.8 Congruence Properties of Bisimulation

We here prove that trace and tensor preserves open maps as promised in Sec. 7.4. This implies that open map bisimulation is a congruence with respect to feedback and parallel composition.

The crucial property we will use is that any colimit-preserving functor between presheaf categories preserves (surjective) open maps [23].

We first show that the trace operator preserves open maps. Recall that the notion of open maps apply canonically to *any* presheaf category, why we also get a notion of open map between presheaves over $(\mathsf{H_n} \times \mathsf{H_p}^{\S})^{\mathsf{op}} \times \mathsf{H_m}$. The desired result then follows from Prop. 8.7.1 if we can show that the functor $\mathcal{S}$ defined in App. 8.7 preserves open maps.

Lemma 8.8.1 *Let $X, Y \colon \mathsf{H_n} \otimes \mathsf{H_p} \to\!\!\!\!\!\bot\!\!\!\!\to \mathsf{H_m} \otimes \mathsf{H_p}$. If $f \colon X \to Y$ is an* open *natural transformation then $\mathcal{S}(f) \colon \mathcal{S}(X) \to \mathcal{S}(Y)$ is open too.*

Proof. Let $f \colon X \to Y$ be an *open* natural transformation. This means that for any arrow $e \colon P \to Q$ of $(\mathsf{H_n} \times \mathsf{H_p}^{\S})^{\mathsf{op}} \times \mathsf{H_m}$ the square

$$\begin{array}{ccc} X(Q) & \xrightarrow{Xe} & X(P) \\ {\scriptstyle f_Q}\downarrow & & \downarrow{\scriptstyle f_P} \\ Y(Q) & \xrightarrow{Ye} & Y(P) \end{array} \tag{8.18}$$

is a quasi-pullback. We must then show that for any arrow $e \colon P \to Q$ of $(\mathsf{H_n} \times \mathsf{H_p}^{\S})^{\mathsf{op}} \times \mathsf{H_m}$ the square

$$\begin{array}{ccc} \mathcal{S}(X)(Q) & \xrightarrow{\mathcal{S}(X)e} & \mathcal{S}(X)(P) \\ {\scriptstyle \mathcal{S}(f)_Q}\downarrow & & \downarrow{\scriptstyle \mathcal{S}(f)_P} \\ \mathcal{S}(Y)(Q) & \xrightarrow{\mathcal{S}(Y)e} & \mathcal{S}(Y)(P) \end{array} \tag{8.19}$$

is a quasi-pullback. It is enough to do this for a class of arrows from which all arrows can be obtained by composition, so it suffices to consider the cases

1. e is an iso, i.e. $e = (1_{\overline{\gamma}}, 1_{[\overline{\alpha},\overline{\beta}]}, 1_{\overline{\delta}}) \colon (\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta}) \to (\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})$,
2. $e = ([\overline{\gamma}, \overline{\gamma}'], 1_{[\overline{\alpha},\overline{\beta}]}, 1_{\overline{\delta}}) \colon (\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta}) \to (\overline{\gamma}', [\overline{\alpha}, \overline{\beta}], \overline{\delta})$, for $\overline{\gamma}' \leq \overline{\gamma}$ in $\mathsf{H_n}$, or
3. $e = (1_{\overline{\gamma}}, 1_{[\overline{\alpha},\overline{\beta}]}, [\overline{\delta}, \overline{\delta}']) \colon (\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta}) \to (\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta}')$, for $\overline{\delta} \leq \overline{\delta}'$ in $\mathsf{H_m}$, or
4. $e = (1_{\overline{\gamma}}, g_{\mathsf{i}}, 1_{\overline{\delta}}) \colon (\overline{\gamma}, [\overline{\beta}, \overline{\beta}], \overline{\delta}) \to (\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})$, for g_{i} the opposite of $[\overline{\alpha}, \overline{\beta}]_{\mathsf{i}} \colon [\overline{\alpha}, \overline{\beta}] \to [\overline{\beta}, \overline{\beta}]$ in $\mathsf{H_p}^{\S}$ or
5. $e = (1_{\overline{\gamma}}, g_{\mathsf{o}}, 1_{\overline{\delta}}) \colon (\overline{\gamma}, [\overline{\alpha}, \overline{\alpha}], \overline{\delta}) \to (\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})$, for g_{o} the opposite of $[\overline{\alpha}, \overline{\beta}]_{\mathsf{o}} \colon [\overline{\alpha}, \overline{\beta}] \to [\overline{\alpha}, \overline{\alpha}]$ in $\mathsf{H_p}^{\S}$.

Case 1 is trivial. Intuitively, case 2 and case 3 express respectively that backtracking of input on ports in $\mathsf{H_n}$ and extension of output on ports in $\mathsf{H_m}$ in $\mathcal{S}(Y)$ can be matched in $\mathcal{S}(X)$. Similarly, case 4 and case 5 express respectively that backtracking of input(communication) on

ports in $\mathsf{H_p}$ and extension of output(communication) on ports in $\mathsf{H_p}$ in $\mathcal{S}(Y)$ can be matched in $\mathcal{S}(X)$.

For the cases 2-5, we use that the diagram 8.18 is a quasi-pullback and use axioms U1, U2 repeatedly. We will only go through case 2, the other cases are treated in a similar fashion.

We want to show that

$$\begin{array}{ccc} \mathcal{S}(X)(\overline{\gamma}', [\overline{\alpha}, \overline{\beta}], \overline{\delta}) & \xrightarrow{\mathcal{S}(X)([\overline{\gamma}', \overline{\gamma}], 1_{[\overline{\alpha}, \overline{\beta}]}, 1_{\overline{\delta}})} & \mathcal{S}(X)(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta}) \\ \downarrow{\scriptstyle \mathcal{S}(f)_{(\overline{\gamma}', [\overline{\alpha}, \overline{\beta}], \overline{\delta})}} & & \downarrow{\scriptstyle \mathcal{S}(f)_{(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})}} \\ \mathcal{S}(Y)(\overline{\gamma}', [\overline{\alpha}, \overline{\beta}], \overline{\delta}) & \xrightarrow{\mathcal{S}(Y)([\overline{\gamma}', \overline{\gamma}], 1_{[\overline{\alpha}, \overline{\beta}]}, 1_{\overline{\delta}})} & \mathcal{S}(Y)(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta}) \end{array} \tag{8.20}$$

is a quasi-pullback. Assume that $x \in \mathcal{S}(X)(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})$, $y \in \mathcal{S}(Y)(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})$ and $y' \in \mathcal{S}(Y)(\overline{\gamma}', [\overline{\alpha}, \overline{\beta}], \overline{\delta})$, such that $\mathcal{S}(f)_{(\overline{\gamma}', [\overline{\alpha}, \overline{\beta}], \overline{\delta})} x = y$ and $\mathcal{S}(Y)([\overline{\gamma}', \overline{\gamma}], 1_{[\overline{\alpha}, \overline{\beta}]}, 1_{\overline{\delta}}) y' = y$. We must then show that there exists $x' \in \mathcal{S}(X)(\overline{\gamma}', [\overline{\alpha}, \overline{\beta}], \overline{\delta})$ such that

$$\mathcal{S}(X)([\overline{\gamma}', \overline{\gamma}], 1_{[\overline{\alpha}, \overline{\beta}]}, 1_{\overline{\delta}}) x' = x \tag{8.21}$$

and

$$\mathcal{S}(f)_{(\overline{\gamma}, [\overline{\alpha}, \overline{\beta}], \overline{\delta})} x' = y'. \tag{8.22}$$

Now, by using the definition of $\mathcal{S}$ the assumption gives us that $x \in X\big((\overline{\gamma}, \overline{\alpha}), (\overline{\beta}, \overline{\delta})\big)$, $y \in Y\big((\overline{\gamma}, \overline{\alpha}), (\overline{\beta}, \overline{\delta})\big)$ and $y' \in Y\big((\overline{\gamma}', \overline{\alpha}), (\overline{\beta}, \overline{\delta})\big)$, such that $f_{(\overline{\gamma}', \overline{\alpha}, \overline{\beta}, \overline{\delta})} x = y$ and $Y\big(([\overline{\gamma}', \overline{\gamma}], 1_{\overline{\alpha}}), (1_{\overline{\beta}}, 1_{\overline{\delta}})\big) y' = y$. Furthermore we have that $w = X\big((1_{\overline{\gamma}}, 1_{\overline{\alpha}}), ([\overline{\beta}, \overline{\alpha}], 1_{\overline{\delta}})\big) x$ is p-secured. By Eq. 8.18 being a quasi-pullback, there exists an $x' \in X\big((\overline{\gamma}', \overline{\alpha}), (\overline{\beta}, \overline{\delta})\big)$ such that $X\big(([\overline{\gamma}', \overline{\gamma}], 1_{\overline{\alpha}}), (1_{\overline{\beta}}, 1_{\overline{\delta}})\big) x' = x$ and $f_{(\overline{\gamma}, \overline{\alpha}, \overline{\beta}, \overline{\delta})} x' = y'$. If we can show that $x' \in \mathcal{S}(X)(\overline{\gamma}', [\overline{\alpha}, \overline{\beta}], \overline{\delta})$ then Eq. 8.22 and Eq. 8.21 follows by definition and the proof is completed. We only need to show that $w' = X\big((1_{\overline{\gamma}'}, 1_{\overline{\alpha}}), ([\overline{\beta}, \overline{\alpha}], 1_{\overline{\delta}})\big) x'$ is p-secured. Now, note that $w \xrightarrow{\mathsf{o}\phi_1} \xrightarrow{\mathsf{o}\phi_2} \cdots \xrightarrow{\mathsf{o}\phi_n} x$ for $\overline{\beta} = \overline{\alpha \phi_1 \phi_2 \dots \phi_n}$, and $x' \xrightarrow{\mathsf{i}\psi_1} \xrightarrow{\mathsf{i}\psi_2} \cdots \xrightarrow{\mathsf{i}\psi_n} x$ for $\overline{\gamma} = \overline{\gamma \psi_1 \psi_2 \dots \psi_n}$. By repeated use of Ax. U2 we get $z \xrightarrow{\mathsf{o}\phi_1} \xrightarrow{\mathsf{o}\phi_2} \cdots \xrightarrow{\mathsf{o}\phi_n} x'$ such that $z \xrightarrow{\mathsf{i}\psi_1} \xrightarrow{\mathsf{i}\psi_2} \cdots \xrightarrow{\mathsf{i}\psi_n} w$. Since w is p-secured it finally follows by repeated use Axioms U1, U2 and A4 that z is p-secured. Finally, since $w' \xrightarrow{\mathsf{o}\phi_1} \xrightarrow{\mathsf{o}\phi_2} \cdot \xrightarrow{\mathsf{o}\phi_n} x'$ it follows by repeated use of Ax. U1 that $w' = z$. □ □

Proposition 8.8.2 *Let $X, Y \colon \mathsf{H_n} \otimes \mathsf{H_p} \multimap \mathsf{H_m} \otimes \mathsf{H_p}$. If $f \colon X \to Y$ is an* open *natural transformation then $\mathsf{Tr}^p_{n,m}(f) \colon \mathsf{Tr}^p_{n,m}(X) \to \mathsf{Tr}^p_{n,m}(Y)$ is open too.*

Proof. Note that $\mathrm{Colim}_{\mathsf{H_p}^{\S}}$ is obviously a colimit preserving functor between presheaf categories, i.e. it preserves open maps. The desired result then follows from Prop. 8.7.1 and Lem. 8.8.1. □

The proof that tensor preserves open maps is simpler than the one for trace, we simply show that tensor can be defined from colimit preserving functors.

Proposition 8.8.3 *Let $X, X' \colon \mathsf{H_{n_1}} \multimap \mathsf{H_{m_1}}$ and $Y, Y' \colon \mathsf{H_{n_2}} \multimap \mathsf{H_{m_2}}$. If $f \colon X \to X'$ and $g \colon Y \to Y'$ are* open *natural transformations then $f \otimes g \colon X \otimes Y \to X' \otimes Y'$ is open too.*

Proof. The tensor of $X\colon \mathsf{H}_{n_1} \multimap \mathsf{H}_{m_1}$ and $Y\colon \mathsf{H}_{n_2} \multimap \mathsf{H}_{m_2}$ can be expressed as a product of presheaves over ${\mathsf{H}_{n_1}}^{\mathsf{op}} \times \mathsf{H}_{m_1} \times {\mathsf{H}_{n_2}}^{\mathsf{op}} \times \mathsf{H}_{m_2}$:

$$X \otimes Y = (\pi_1^* X) \times (\pi_2^* Y)$$

where e.g.

$$\pi_1^*\colon [\mathsf{H}_{n_1} \times {\mathsf{H}_{m_1}}^{\mathsf{op}}, \mathsf{Set}] \to [\mathsf{H}_{n_1} \times {\mathsf{H}_{m_1}}^{\mathsf{op}} \times \mathsf{H}_{n_2} \times {\mathsf{H}_{m_2}}^{\mathsf{op}}, \mathsf{Set}]$$

is obtained by composition with the projection

$$\pi_1\colon \mathsf{H}_{n_1} \times {\mathsf{H}_{m_1}}^{\mathsf{op}} \times \mathsf{H}_{n_2} \times {\mathsf{H}_{m_2}}^{\mathsf{op}} \to \mathsf{H}_{n_1} \times {\mathsf{H}_{m_1}}^{\mathsf{op}},$$

so $\pi_1^*(X)(\overline{\alpha}_1, \overline{\beta}_1)(\overline{\alpha}_2, \overline{\beta}_2) = X(\overline{\alpha}_1, \overline{\beta}_1)$. For general reasons π_1^* has a right adjoint (constructed as a right Kan extension - see [82, 23]). Thus π_1^* and, similarly, π_2^* are left adjoints and so preserve (surjective) open maps. Combined with the similar fact about product of presheaves we deduce that $\otimes$ preserves (surjective) open maps. □

Bibliography

[1] Samson Abramsky. On semantic foundations for applicative multiprogramming. In J. Diaz, editor, *Automata, Languages and Programming*, volume 154 of *LNCS*, pages 1–14. Springer-Verlag, July 1983.

[2] Samson Abramsky. A generalized kahn principle for abstract asynchronous networks. In *MFPS'89*, volume 442 of *LNCS*, pages 1–21. Springer, 1990.

[3] Samson Abramsky. Retracing some paths in process algebra. In *CONCUR'96*, volume 1119 of *LNCS*, pages 1–17, 1996.

[4] Samson Abramsky, Simon Gay, and Rajagopal Nagarajan. Interaction categories and the foundations of typed concurrent programming. In *Proc. of the 1994 Marktoberdorf summer school*. Springer, 1994.

[5] Samson Abramsky, Simon Gay, and Rajagopal Nagarajan. Specification structures and propositions-as-types for concurrency. In F. Moller and G. Birtwistle, editors, *Proc. of the VIIIth Banff Higher Order Workshop*, LNCS. Springer, 1995.

[6] Samson Abramsky and Radha Jagadeesan. New foundations for the geometry of interaction. *Information and Computation*, 111(1):53–119, May 1994.

[7] Peter Aczel. A semantic universe for fairness. Preliminary Draft, 1996.

[8] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[9] E. S. Bainbridge. Feedback and generalized logic. *Information and Control*, 31:75–96, 1976.

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

[11] M. A. Bednarczyk. *Categories of asynchronous systems*. PhD thesis, University of Sussex, 1988.

[12] M. A. Bednarczyk. Hereditary history preserving bisimulation or what is the power of the future perfect in program logics. Technical report, Polish Acad. of Sciences, Gdansk, 1991.

[13] J. Benabou. Fibred categoires and the foundations of naive category theory. *Journal of Symbolic Logic*, 50:10–37, 1985.

[14] Eike Best, Raymond Devillers, Astrid Kiehn, and Lucia Pomello. Concurrent bisimulations in Petri nets. *Acta Inform.*, 28(3):231–264, 1991.

[15] F. Borceux. *Handbook of categorical logic*, volume 1. Cambridge University Press, 1994.

[16] Jarvis Dean Brock. *A Formal Model of Non-determinate Dataflow Computation*. PhD thesis, Laboratory for Computer Science, MIT, 1983.

[17] J.D. Brock and W.B. Ackerman. Scenarios: A model of non-determinate computation. In J. Diaz and I Ramos, editors, *Formalization of Programming Concepts*, volume 107 of *LNCS*. Springer, 1981.

[18] Gian Luca Cattani, Marcelo Fiore, and Glynn Winskel. A theory of recursive domains with applications to concurrency. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science*, 1998.

[19] Gian Luca Cattani, Andrew John Power, and Glynn Winskel. A categorical axiomatics for bisimulation. In *Proceedings of the 9th International Conference on Concurrency Theory, CONCUR '98*, volume 1466 of *LNCS*, pages 581–596. Springer-Verlag, 1998.

[20] Gian Luca Cattani and Vladimiro Sassone. Higher dimensional transition systems. In *LICS'96*. IEEE, 1996.

[21] Gian Luca Cattani, Ian Stark, and Glynn Winskel. Presheaf models for the pi-calculu. In *CTCS'97*, volume 1290 of *LNCS*, pages 106–126. Springer, 1997.

[22] Gian Luca Cattani and Glynn Winskel. Presheaf models for concurrency. In *CSL'96*, volume 1258 of *LNCS*, pages 58–75. Springer, 1997.

[23] Gian Luca Cattani and Glynn Winskel. Presheaf models for concurrency. In *Computer Science Logic: Tenth international Workshop, CSL'96, Annual Conference of the EACSL. Selected Papers*, number 1258 in Lecture Notes in Computer Science, pages 58–75. Springer-Verlag, 1997.

[24] Gina Luca Cattani. *Presheaf models for concurrency*. PhD thesis, Aarhus University, 1999.

[25] K. Ĉerans. Decidability of bisimulation equivalence for parallel timed processes. In *Proceedings of CAV'92*, volume 663 of *LNCS*, 1992.

[26] Allan Cheng. Petri nets, traces, and local model checking. Technical Report RS-95-39, BRICS, 1995.

[27] Allan Cheng. *Reasoning About Concurrent Computational Systems*. Dissertation series, BRICS, Department of Computer Science, University of Aarhus, August 1996. Ph.D. thesis. xiv+229 pp.

[28] Allan Cheng and Mogens Nielsen. Open maps (at) work. Research Series RS-95-23, BRICS, Department of Computer Science, University of Aarhus, April 1995.

[29] Clausen. Christian. Bisimulation, games, and logic. Master's thesis, Aarhus University, December 1993.

[30] Alonzo Church. An unsolvable problem of elementary number theory. *American J. Math.*, 58:345–363, 1936.

[31] Roy L. Crole. *Categories for Types*. CUP, 1993.

[32] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. Partial orderings descriptions and observations of nondeterministic concurrent processes. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 438–466. Springer, 89.

[33] Jack B. Dennis. First version of a dataflow procedure language. In B. Robinet, editor, *Proceedings Colloque sur la Programmation*, volume 19 of *LNCS*, pages 362–376. Springer, 1974.

[34] V. Diekert and Y. Métivier. *Handbook of Formal Languages.*, chapter Partial Commutation and Traces. Springer, 1997.

[35] Lisbeth Fajstrup, Eric Goubault, and Martin Raussen. Algebraic topology and concurrency. Technical Report R-99-2008, Department of Mathematical Sciences, Aalborg University, June 1999.

[36] A. A. Faustini. An operational semantics for pure dataflow. In *ICALP'82*, volume 140 of *LNCS*, pages 212–224. Springer, 1982.

[37] Marcelo Fiore, Gian Luca Cattani, and Glynn Winskel. Weak bisimulation and open maps. In *Proceedings of the Fourteenth Annual IEEE Symposium on Logic in Computer Science*, 1999.

[38] Marcelo P. Fiore. A coinduction principle for recursive data types based on bisimulation. *Information and Computation*, 127, 1996.

[39] N. Francez. *Fairness*. Springer-Verlag, 1986.

[40] Sibylle Fröschle. On the decidability problem of strong history-preserving bisimulation. Progress Report, 1998.

[41] Sibylle Fröschle. Decidability of plain and hereditary history-preserving bisimulation for bpp. In *Proceedings of the 6th International Workshop on Expressiveness in Concurrency*, number 27 in ENTCS, 1999.

[42] Sibylle Fröschle and Thomas Troels Hildebrandt. On plain and hereditary history-preserving bisimulation. Technical Report RS-99-4, BRICS, Department of Computer Science, University of Aarhus, 1999.

[43] Simon John Gay. *Linear Types for Communicating Processes*. PhD thesis, University of London, Imperial College of Science, Technology and Medicine, Department of Computing, january 1995.

[44] J.-Y. Girard. Geometry of interaction I: interpretation of system F. In Ferro et. al., editor, *Proceedings Logic Colloquium 88*, pages 221–260. North-Holland, 1989.

[45] J.-Y. Girard. Geometry of interaction II: deadlock free algorithms. In P. Martin-Lof and G. Mints, editors, *Proceedings of COLOG 88*, number 417 in Lecture Notes In Computer Science, pages 76–93. Springer-Verlag, 1989.

[46] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *CAV'91*, volume 575 of *LNCS*, pages 332–342. Springer, 1991.

[47] Patrice Godefroid. On the costs and benefits of using partial-order methods for the verification of concurrent systems. In *POMIV'96*, pages 289–304, 1996.

[48] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*. Springer, 1996.

[49] Mike Gordon. The semantic challenge of verilog hdl. www.cl.cam.ac.uk/mjcg/, April 1996. Revised version of an invited paper published in LICS'95.

[50] Eric Goubault. Domains of higher-dimensional automata. In *CONCUR'93*, volume 715 of *LNCS*, pages 293–307. Springer, 1993.

[51] Eric Goubault and T. P. Jensen. Homology of higher-dimensional automata. In *CONCUR'92*, volume 630 of *LNCS*, pages 254–268. Springer, 1992.

[52] Masahito Hasegawa. Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. In *TLCA'97*, volume 1210 of *LNCS*, pages 196–213, 1997.

[53] Mathew Hennessy. Modelling finite delay operators. Technical report, University of Edinburgh, 1983.

[54] Mathew Hennessy and Colin Stirling. The power of the future perfect in program logics. *Information and Control*, pages 23–52, 1985.

[55] Matthew Hennessy. *Algebraic theory of processes*. Series in the Foundations of Computing. MIT Press, 1988.

[56] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.

[57] Thomas Troels Hildebrandt. Extensions of independence models. Progress report, May 1997.

[58] Thomas Troels Hildebrandt. A fully abstract presheaf semantics of SCCS with finite delay. Research Series RS-99-28, BRICS, Department of Computer Science, University of Aarhus, September 1999. To appear in Proceedings of CTCS'99.

[59] Thomas Troels Hildebrandt, Prakash Panangaden, and Glynn Winskel. A relational model of non-deterministic dataflow. In *CONCUR'98*, volume 1466 of *LNCS*, pages 613–628. Springer-Verlag, 1998.

[60] Thomas Troels Hildebrandt and Vladimiro Sassone. Comparing transition systems with independence and asynchronous transition systems. Research Series RS-96-18, BRICS, Department of Computer Science, University of Aarhus, June 1996. 14 pp. Appears in proceedings of CONCUR'96, LNCS 1119, pages 84–97.

[61] Thomas Troels Hildebrandt and Vladimiro Sassone. Transition systems with independence and multi-arcs. Research Series RS-97-10, BRICS, Department of Computer Science, University of Aarhus, 1997. Appears in Procedings of POMIV'96, DIMACS vol. 29, pp. 273-288.

[62] C. A. R. Hoare. *Communicating sequential processes.* Series in Computer Science. Prentice-Hall, 1985.

[63] Carl A. R. Hoare. A model for communicating sequential processes. Technical Report PRG-22, Programming Research Group, University of Oxford Computing Lab, 1981.

[64] Michael Huth and Marta Kwiatkowska. The semantics of fair recursion with divergence. Submitted. Technical report CSR-96-7.

[65] Michael Huth and Marta Kwiatkowska. Finite but unbounded delay in synchronous CCS. In A. Edalat, S. Jourdan, and G. McCusker, editors, *Advanced methods in theory and formal methods of computing: Proceedings of the third Imperial College workshop April 1996*, pages 312–323. Imperial College Press, 1996.

[66] Lalita Jategaonkar and Albert R. Meyer. Deciding true concurrency equivalences on finite safe nets. In *ICALP'93*, volume 700. Springer, 1993. Full version in TCS 154:107-143,1996.

[67] Vaughan F.R. Jones. A polynomial invariant for links via von neumann algebras. *Bull. Amer. Math. Soc.*, 129:103–112, 1985.

[68] Bengt Jonsson. A fully abstract trace model for dataflow networks. In *POPL'89*, pages 155–165. ACM, 1989.

[69] A Joyal and I Moerdijk. A completeness theorem for open maps. *Annals of Pure and Applied Logic*, 70(1):51–86, 1994.

[70] Andr Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. In *Math. Proc. Camb. Phil. Soc.*, volume 119, pages 447–468, 1996.

[71] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *LICS '93 special issue of Information and Computation*, 127(2):164–185, juni 1996. Available as BRICS report, RS-94-7.

[72] Marcin Jurdziński and Mogens Nielsen. Hereditary history preserving bisimilarity is undecidable. Research Series RS-99-19, BRICS, Department of Computer Science, University of Aarhus, June 1999. 18 pp.

[73] G. Kahn and D. MacQueen. Coroutines and networks of parallel processes. In Gilchrist, editor, *Proceedings of Information Processing*, pages 993–998. North-Holland, 1977.

[74] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processeing*, volume 74, pages 471–475, 1974.

[75] P. Kannelakis and S. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[76] P Katis, N Sabadini, and RFC Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 1997.

[77] R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, July 1976.

[78] J. Kok. A fully abstract semantics for dataflow nets. In *Proceedings of Parallel Architectures And Languages Europe*, pages 351–368, Berlin, 1987. Springer.

[79] Leslie Lamport and Nancy Lynch. Distributed computing models and methods. In Jan van Leeuven, editor, *Handbook of Theoretical Computer Science*, volume 2, chapter 18, pages 1157–1199. Elsevier Science B.V., 1990.

[80] K. Larsen and A. Skou. Bisimulation through probailistic testing. In *Principles of Programming Languages*, pages 344–351, 1989.

[81] Nancy A. Lynch and Eugene W. Stark. A proof of the kahn principle for input/output automata. *Information and Computation*, 82:81–92, 1989.

[82] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

[83] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer, 1992.

[84] Jerome Malitz. *Introduction to Mathematical Logic*. Undergraduate Texts in Mathematics. Springer-Verlag, 1979.

[85] Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report PB-78, DAIMI, Computer Science Department, University of Aarhus, 1977.

[86] Antoni Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 279–324. Springer, 1986.

[87] Robin Milner. A calculus of communicating systems, 1980.

[88] Robin Milner. A finite delay operator in synchronous ccs. Technical report, University of Edinburgh, Dept. of Computer Science, Kings Buildings, 1982.

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

[90] Robin Milner. *Communication and Concurrency*. Series in Computer Science. Prentice Hall, 1989.

[91] Robin Milner. The polyadic π-calculus: a tutorial. In *Logic and algebra of specification (Marktoberdorf, 1991)*, pages 203–246. Springer, 1993.

[92] Ugo Montanari and Marco Pistore. Minimal transition systems for history-preserving bisimulation. In *STACS '97*, volume 1200 of *LNCS*, pages 413–425. Springer, 1997.

[93] Madhavan Mukund and Mogens Nielsen. Ccs, locations and asynchronous transition systems. In R Shyamasundar, editor, *Proceedings of FST & TCS'92*, number 652 in LNCS, pages 328–341. Springer, 1992.

[94] Madhavan Mukund and P. S. Thiagarajan. Linear time temporal logics over Mazurkiewicz traces. In *POMIV '96*, volume 29 of *DIMACS*, pages 171–201. AMS, 1997.

[95] Mogens Nielsen and Christian Clausen. Bisimulations, games and logic. Research Series RS-94-6, BRICS, Department of Computer Science, University of Aarhus, April 1994. 37 pp. Full version of paper appearing in: *New Results and Trends in Computer Science*, pages 289–305, LNCS 812, 1994.

[96] Mogens Nielsen and Christian Clausen. Games and logics for a noninterleaving bisimulation. *Nordic Journal of Computing*, 2(2):221–249, 1995.

[97] Mogens Nielsen and Thomas S. Hune. Timed bisimulation and open maps. Research Series RS-98-4, BRICS, Department of Computer Science, University of Aarhus, February 1998. 27 pp. Appears in LNCS 1450, pages 378–387.

[98] Mogens Nielsen and Marcin Jurdziński. Hereditary history preserving simulation is undecidable. Research Series RS-99-1, BRICS, 1999.

[99] Mogens Nielsen, G. Rozenberg, and P. S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96:3–33, 1992.

[100] Mogens Nielsen and Glynn Winskel. Petri nets and bisimulation. *TCS*, 153:211–244, 1996.

[101] Prakash Panangaden. The expressive power of indeterminate primitives in asynchronous computations. Technical Report SOCS-95.8, School of CS, McGill, 1995.

[102] Prakash Panangaden and Vasant Shanbhogue. The expressive power of indeterminate dataflow primitives. *Information and Computation*, 98(1):99–131, May 1992.

[103] Prakash Panangaden and Eugene W. Stark. Computations, residuals and the power of indeterminacy. In *Proc. of the 15th ICALP*, pages 439–454. Springer, 1988.

[104] D.M.R Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science: 5th GI-Conference*, volume 104 of *LNCS*, pages 168–183. Springer, 1981.

[105] D. Peled. All from one, one for all: on model checking using representatives. In *CAV'93*, volume 697 of *LNCS*, pages 409–423. Springer, 1993.

[106] C.A. Petri. Non-sequential processes. Technical Report ISF-77-05, GMD-ISF, 1977.

[107] Gordon Plotkin. A structural approach to operational semantics. DAIMI FN - 19, Computer Science Department, Aarhus University, September 1981.

[108] Gordon Plotkin. A powerdomain for countable non-determinism. In *Automata, Languages and Programming (ICALP), Ninth Colloquium*, volume 140 of *LNCS*, pages 418–428. Springer-Verlag, 1982.

[109] Amir Pnueli. Applications of temporal logic to the specifiaction and verification of reactive systems: a survey of current trends. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Current Trends in Concurrency*, volume 224 of *LNCS*, pages 1–50. Springer-Verlag, 1985.

[110] Vaughan R. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, february 1986.

[111] Vaughan R. Pratt. Modelling concurrency with geometry. In *Proc. of the 18th POPL*, pages 311–322. ACM, 1991.

[112] A. Rabinovich and B. A. Trakhtenbrot. Nets of processes and dataflow. To appear in Proceedings of ReX School on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS, 1988.

[113] A. Rabinovich and B. A. Trakhtenbrot. Nets and data flow interpreters. In *Proceedings of the 4th LICS*, pages 164–174, 1989.

[114] A. Rabinovich and B. A. Trakhtenbrot. Communication among relations. In M. S. Paterson, editor, *Proc. of the 7th ICALP*, volume 443 of *LNCS*, pages 294–307. Springer, 1990.

[115] A. Rabinovich and B.A. Trakhtenbrot. Behaviour structures and nets. *Fundamenta Informaticae*, XI(4):357–404, 1988.

[116] James R. Russell. Full abstraction for nondeterministic dataflow networks. In *FOCS'89*, pages 170–176, 1989.

[117] Jan Rutten and Daniele Turi. Initial algebra and final coalgebra semantics for concurrency. In J. de Bakker et al., editor, *Proceedings of the REX workshop A Decade of Concurrency - Reflections and Perspectives*, volume 803 of *LNCS*, pages 530–582. Springer-Verlag, 1994.

[118] Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: Towards a classification. *To appear in Theoretical Computer Science*, 1993.

[119] Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Relationships between models for concurrency. DAIMI PB 456, University of Aarhus, December 1993.

[120] Peter Selinger. First-order axioms for asynchrony. In Antoni Mazurkiewicz and Jzef Winkowski, editors, *CONCUR'97*, volume 1243 of *LNCS*, pages 376–390. Springer, 1997.

[121] Peter Selinger. Categorical structure of asynchrony. *Electronic Notes in Theoretical Computer Science*, 20, 1999.

[122] M. W. Shields. Concurrent machines. *Computer Journal*, 28:449–465, 1985.

[123] John Staples and V. L. Nguyen. A fixpoint semantics for nondeterministic data flow. *Journal of the ACM*, 32(2):411–444, April 1985.

[124] Eugene W. Stark. Compositional relational semantics for indeterminate dataflow networks. In *CTCS*, volume 389 of *LNCS*, pages 52–74, Manchester, U.K., 1989. Springer.

[125] Eugene W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 63-64:221–269, 1989.

[126] Eugene W. Stark. Dataflow networks are fibrations. Technical report, Dept. of Computer Science, Stony Brook, 1989.

[127] Eugene W. Stark. A calculus of dataflow networks. In *LICS '92, Proceedings of the 7th Annual Symposium on Logic in Computer Science*, pages 125–136. IEEE Computer Society Press, June 1992.

[128] Ross Street. Fibrations in bicategories. *Cahiers de topologie et gomtrie diffrentielle*, XXI(2 - 2^e trimestre):111–160, 1980.

[129] A. Valmari. A stubborn attack on state explosion. In *CAV'90*, number 3 in DIMACS series, pages 25–41, 1991.

[130] Rob J. van Glabbeek. Bisimulations for higher dimensional automata. E-mail message sent to the Concurrence mailing list on July 7, 1991. Available at http://theory.stanford.edu/rvg/hda.

[131] Rob J. van Glabbeek and Ursula Goltz. Equivlaence notions for concurrent systems and refinement of actions. In *MFCS'89*, volume 379 of *LNCS*. Springer, 1989.

[132] Rob J. van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems, 1998. http://theory.stanford.edu/ rvg/abstracts.html.

[133] Rob J. van Glabbeek and Gordon Plotkin. Configuration structures. In *LICS'95*, pages 99–109. IEEE, 1995.

[134] Walter Vogler. Deciding history preserving bisimulation. In *ICALP'91*, volume 510 of *LNCS*, pages 495–505, 1991.

[135] Walter Vogler. Bisimulation and action refinement. *Theor. Comp. Sci.*, 114(1):173–200, 1993.

[136] Walter Vogler. Generalized OM-bisimulation. *Inform. and Comput.*, 118(1):38–47, 1995.

[137] Glynn Winskel. *Events in Computation.* PhD thesis, University of Edinburgh, 1980.

[138] Glynn Winskel. Event structure semantics of ccs and related languages. In Mogens Nielsen and Erik M. Schmidt, editors, *Icalp '82*, volume 140 of *LNCS*, pages 561–576. Springer-Verlag, 1982.

[139] Glynn Winskel. Generalised synchronisation trees. Handwritten notes, 1983.

[140] Glynn Winskel. Category theory and models for parallel computation. In David Pitt, Samson Abramsky, Axel Poign, and David Rydeheard, editors, *Category Theory and Computer Programming*, volume 240 of *LNCS*, pages 266–281. Springer-Verlag, September 1985.

[141] Glynn Winskel. Synchronisation trees. *Theoretical Computer Science*, 34:33–82, 1985.

[142] Glynn Winskel. Event structures. In Brauer, Reissig, and Rozenberg, editors, *Petri Nets: Applications and relationships to other models of concurrency*, number 255 in LNCS, pages 325–392. Springer-Verlag, 1987.

[143] Glynn Winskel. A presheaf semantics of value-passing processes. In *CONCUR'96*, volume 1119 of *LNCS*, pages 98–114, 1996.

[144] Glynn Winskel. A linear metalanguage for concurrency. In *AMAST '98*, volume 1548 of *LNCS*, pages 42–58. Springer-Verlag, 1998.

[145] Glynn Winskel. Event structures as presheaves—two representation theorems. In *CONCUR'99*, 1999. Appears as a BRICS technical report RS-99-7.

[146] Glynn Winskel and Mogens Nielsen. *Handbook of Logic in Computer Science*, volume IV, chapter Models for concurrency. OUP, 1995.

[147] Glynn Winskel and Mogens Nielsen. Presheaves as transition systems. In Doron Peled, Vaughan Pratt, and Gerard Holzmann, editors, *POMIV'96*, volume 29 of *DIMACS*. AMS, july 1996.

[148] Oswald Wyler. *Lecture Notes on Topoi and Quasitopoi*. World Scientific, 1991.

Recent BRICS Dissertation Series Publications

DS-00-1 **Thomas Troels Hildebrandt.** ***Categorical Models for Concurrency: Independence, Fairness and Dataflow*****. February 2000. PhD thesis. x+141 pp.**

DS-99-1 **Gian Luca Cattani.** ***Presheaf Models for Concurrency*****. April 1999. PhD thesis. xiv+255 pp.**

DS-98-3 **Kim Sunesen.** ***Reasoning about Reactive Systems*****. December 1998. PhD thesis. xvi+204 pp.**

DS-98-2 **Søren B. Lassen.** ***Relational Reasoning about Functions and Nondeterminism*****. December 1998. PhD thesis. x+126 pp.**

DS-98-1 **Ole I. Hougaard.** ***The CLP(OIH) Language*****. February 1998. PhD thesis. xii+187 pp.**

DS-97-3 **Thore Husfeldt.** ***Dynamic Computation*****. December 1997. PhD thesis. 90 pp.**

DS-97-2 **Peter Ørbæk.** ***Trust and Dependence Analysis*****. July 1997. PhD thesis. x+175 pp.**

DS-97-1 **Gerth Stølting Brodal.** ***Worst Case Efficient Data Structures*****. January 1997. PhD thesis. x+121 pp.**

DS-96-4 **Torben Braüner.** ***An Axiomatic Approach to Adequacy*****. November 1996. Ph.D. thesis. 168 pp.**

DS-96-3 **Lars Arge.** ***Efficient External-Memory Data Structures and Applications*****. August 1996. Ph.D. thesis. xii+169 pp.**

DS-96-2 **Allan Cheng.** ***Reasoning About Concurrent Computational Systems*****. August 1996. Ph.D. thesis. xiv+229 pp.**

DS-96-1 **Urban Engberg.** ***Reasoning in the Temporal Logic of Actions — The design and implementation of an interactive computer system*****. August 1996. Ph.D. thesis. xvi+222 pp.**