Fundamental Study

An extensional fixed-point semantics for nondeterministic data flow

Peter Kearney*

School of Computing and Information Technology, Griffith University, Nathan, Brisbane, 4111, Australia

John Staples

Department of Computer Science, University of Queensland, St. Lucia, Brisbane, 4072, Australia

Communicated by D. Park
Received February 1987
Revised April 1988

Abstract


A fixed point semantics for nondeterministic data flow is introduced which refines and extends work of Park (1983). It can be seen also as an extension to the general case of Kahn's (1974) successful fixed point semantics for deterministic data flow. An associativity result for network construction is proved which shows that anomalies such as those of Brock and Ackerman do not arise in this semantics. The semantics is shown to be extensional, in the natural sense that nondeterministic processes which induce identical input–output relations in all contexts are equal.

Contents

1. Introduction .............................................. 130
2. Basic domains ........................................... 135
   2.1. Data domains ....................................... 135
   2.2. Oracle values and oracle sequences .............. 136
   2.3. A partial order for oracle sequences .......... 139

*Current address: Department of Computer Science, University of Queensland, St. Lucia, Brisbane, 4072, Australia.

0304-3975/91/$03.50 © 1991—Elsevier Science Publishers B.V. All rights reserved
1. Introduction

Kahn [5] presented a successful fixed-point semantics for deterministic data flow computation. It is well known that this semantics does not extend readily to nondeterministic data flow computation. There have been many suggestions for a fixed point semantics of nondeterministic data flow, e.g. [3, 8, 10, 12, 13]. This article presents a semantics in the style of, but refining and extending, the ideas presented by Park in [10]. For an introduction to data flow networks, a review of Kahn’s model and a presentation of problems in modelling nondeterminism, see the introductory sections of [13].
The primary motivation for the semantics of this paper can be described as scientific rather than philosophical. That is, the aim is to develop a model of nondeterministic data flow which conveniently supports deductive reasoning and computational experiments. In particular, we aim to retain as much as possible of the convenience of Kahn's analysis of deterministic data flow. Our semantics is not primarily intended to answer philosophical questions about the ultimate nature of nondeterminism, though if it is successful in its scientific objectives, it will indirectly contribute in this direction also.

The semantics presented here is based on a view of nondeterminism as arising from a lack of information. We will regard nondeterministic processes as manifestations of underlying deterministic processes. In fact, to reflect an observer's limited knowledge about the underlying deterministic process, each nondeterministic process will be modelled by an equivalence class of deterministic processes. Nevertheless, we shall often refer loosely to members of the equivalence classes as deterministic models.

The relationship of our work to operational semantics of nondeterministic computation is as follows. Models of machine behaviour are described as operational when they are believed to correspond to the behaviour of actual or possible machines. There is no agreed standard operational semantics for nondeterministic computation. Our approach is based on Kahn's model for deterministic data flow, which is widely agreed to admit a satisfactory operational semantics (see e.g. [4]). This provides a basis for relating our work to operational behaviours. Operational views of our modelling of time delays and fair nondeterministic behaviour are discussed below.

Here we investigate systems which can include an unbounded number of sources of fair nondeterministic behaviour. It is widely agreed that nondeterminism is required to model various aspects of real-system behaviour. It also has the potential to act as an information hiding mechanism for simplifying the modelling of complex deterministic behaviours.

The occurrence of fair sources of nondeterministic behaviour in physical reality is open to dispute. The reality of computing systems, however, is that operator intervention is typically used to eliminate unfairness. For example, random delays may occur on an unreliable communications channel, but it can reasonably be assumed that an infinite delay would be excluded by operator intervention. The channel would be diagnosed as faulty, and repaired or replaced.

On a given run, a nondeterministic process makes a number of choices which are not predictable or controllable. Our model uses extra parameters at the deterministic level to allow deterministic methods of analysis of such runs. Nondeterministic behaviours are obtained by abstracting from the deterministic level. The present approach is quite different from the approaches (e.g. [1, 11]) which seek to construct convenient mathematical structures at the nondeterministic level, without recourse to extra parameters and a deterministic framework. We believe that the relative merits of the different approaches need to be assessed by applications of the semantics to the specification and verification of practical nondeterministic processes. We hypothesise that our approach will provide a convenient framework for practical specification and
verification. Some evidence is available to support this hypothesis [6], but much more work needs to be done.

For a given nondeterministic process, one of its deterministic models is used to calculate nondeterministic input–output behaviours as follows. For a given input history of the nondeterministic process an arbitrary choice is made of the extra information used in the deterministic model. The output history for the deterministic model is then calculated using Kahn's semantics. The output history includes information significant for the deterministic model but not for the nondeterministic process. This extra information in the deterministic output history is discarded to obtain the output history of the nondeterministic process. By ranging over the possible choices of extra input information we can calculate the behaviours of the nondeterministic process. There are two aspects of the above method: the extra information introduced and the arbitrary choice of this information. The arbitrary choices which are characteristic of nondeterministic behaviour are provided for in our semantics by equipping each deterministic modelling process with a single oracle input, which does not appear at the nondeterministic level. The restriction to a single oracle, which is not part of Park's approach, enhances the modularity of the theory and does not limit its power.

The extra information used to determine the behaviour of a nondeterministic process may be motivated as follows. In a deterministic data flow network the data in different streams are implicitly synchronised (logically rather than in physical time) by the positions of the data in the streams. In a nondeterministic process this sort of synchronisation is often inappropriate. For example a nondeterministic merge does not necessarily make the items at the head of its two input streams adjacent in its output stream. To avoid this implicit synchronisation we introduce a new data item, here called "wait" (the "hiaton" of [lo], which cites [14]). This wait data item is used in the following two ways.

1. Finitely many occurrences of wait may appear between "real" (that is, nonwait) data items. Such occurrences can be regarded as desynchronising data streams. These interpolations can be made by individual deterministic modelling processes, as illustrated in the following example. Such an interpolation is also made arbitrarily when modelling a nondeterministic data stream by a deterministic data stream.

2. Data streams which are finite at the nondeterministic level are extended to infinite data streams in a deterministic model by appending an infinite sequence of waits. In the case of external inputs to a deterministic network model, this extension is part of the translation from the nondeterministic level to the deterministic level. For all other data streams, i.e. internal streams and network outputs, our deterministic models achieve this extension automatically.

Waits may be interpreted operationally as terminating a segment of data transmitted at a given physical time. (We will be more precise about this in Section 3). In this interpretation the behaviours of our modelling processes may be sensitive to time delays in inputs. However, this time sensitivity is modelled by nondeterministic generation of waits, not by any synchronisation mechanism. Our processes are
asynchronous. There is no high-level synchronisation in the sense that there is no implicit reverse control flow along data lines due to a receiver deciding to synchronise with a transmitter.

**Example.** The “fair” nondeterministic merge is a process with two input streams and one output stream which merges *all* its input data onto the output stream. Our basic intention is to model this process by a simple deterministic merge (say an alternating merge) of suitable deterministic models of the input histories. By arranging that the data streams for the deterministic merge are infinite, we ensure that the deterministic merge does not deadlock on either input, leaving data unpassed on the other input. For further discussion of this point, see [10].

To model the variety of choices which is available to a fair nondeterministic merge, our deterministic model also uses its oracle input to insert arbitrary interpolations of waits into the input streams to the deterministic merge, as indicated in Fig. 1. It also illustrates the distribution of the network's oracle input to modules which require their own oracle input. For this approach to succeed in modelling fair merge it is necessary that the wait injectors depicted in Fig. 1 inject only finitely many waits between any two “real” data items, or else a real data item may be neglected forever, causing unfairness. To ensure fairness, we impose the global requirement that oracle inputs be fair, in the strong sense conveyed by Definition 2.2.13. Because our

![Diagram of a deterministic model for a fair nondeterministic merge](image)

Fig. 1. Sketch of a deterministic model for a fair nondeterministic merge.
semantics always uses strongly fair oracle inputs, the oracle distributor and wait
injector modules can be simple in design. Note that strong fairness of oracle sequences
is not a restriction on our theory. We shall show in Section 6.3 that all fair and unfair
sequences can be generated by suitable modelling processes. Then, modelling pro-
cesses may be driven by such sequences. Here are suitable definitions for all the
deterministic modules depicted in Fig. 1.

Oracle sequences are infinite sequences of 0's and 1's. Writing \( s \) for the input stream
of the oracle distributor, its left and right output streams \( \lambda s, \rho s \) are oracle sequences
defined by:

\[
(\lambda s)(n) = s(2n - 1) \quad n \geq 1.
\]

\[
(\rho s)(n) = s(2n) \quad n \geq 1.
\]

We shall subsequently refer to this oracle distributor as Dist.

A suitable wait injector, inject, may be defined as follows, where the first argument
is the data stream and the second argument is the oracle input. We denote by \( X . Y \) the
concatenation of the sequences \( X \) and \( Y \). The symbol \( \bot \) denotes the empty sequence
and the symbol \( \tau \) denotes the wait data item.

\[
\text{inject}(X, s) = \tau \cdot \text{inject}'(X, s).
\]

\[
\text{inject}'(X, \bot) = \bot.
\]

\[
\text{inject}'(\bot, s) = \bot.
\]

\[
\text{inject}'(a.X, 0.s) = a \cdot \text{inject}'(X, s).
\]

\[
\text{inject}'(X, 1.s) = \tau \cdot \text{inject}'(X, s).
\]

An alternating deterministic merge, dmerge, may be defined as follows.

\[
\text{dmerge}(\bot, Y) = \tau.
\]

\[
\text{dmerge}(a.X, Y) = \tau . a . \text{dmerge}'(X, Y).
\]

\[
\text{dmerge}'(X, \bot) = \tau.
\]

\[
\text{dmerge}'(X, b.Y) = \tau . b . \text{dmerge}(X, Y).
\]

These functions produce an "initial" wait. This is an example of a general require-
ment, which will be introduced in Section 3.

Finally, Fig. 1 also illustrates that we may neglect to mention oracle inputs of
modules, such as deterministic merge, which do not make any use of them. In theory,
however, for the sake of uniformity, every module is regarded as having an oracle
input.

Given the deterministic models, our basic intention is to calculate the input–output
behaviours of a nondeterministic process by ranging over possible oracle inputs to the
deterministic model and by ignoring waits in data streams. This approach does not lead naively to an extensional theory. That is, two different deterministic models may behave in all contexts in ways which are indistinguishable at the nondeterministic level. To achieve extensionality, we abstract from the precise response of any one deterministic model to a given oracle input. We define an equivalence relation ("behaviour equivalence") on deterministic processes. Then our model of a nondeterministic network will be a behaviour equivalence class of deterministic processes. In Section 2 we define the domains used in our deterministic models. The class of strongly fair oracle sequences is defined and a partial order is defined on the strongly fair oracle sequences. In Section 3 we define the deterministic processes used in deterministic models. These deterministic modelling processes map infinite input sequences to infinite output sequences.

In Section 4 we introduce the operators used to model network construction. We show that such modelling operators map modelling processes to modelling processes and that the class of modelling processes is closed under recursion using modelling operators. A specific useful class of modelling operators is introduced, which is used for modelling network constructions. We prove an associativity result for deterministic network construction. In Section 5 we define the behaviour equivalence relation over deterministic processes and show that behaviour equivalence is preserved under network construction. In Section 6 we define the translation from the nondeterministic level of the theory to the deterministic level. We show associativity of nondeterministic network construction. We also show that our modelling of nondeterministic data flow processes is extensional in the well-defined theoretical sense that nondeterministic processes which induce identical input-output relations in all contexts are equal. Put differently, the extensionality result shows that if processes are not identical in our semantics, the difference will result in different nondeterministic input-output behaviour in some context. In Section 7 we outline an alternative metric space framework for analysing modelling networks. In Section 8 we sum up, and compare this work with [10].

2. Basic domains

2.1. Data domains

In general, the structure of data domains in our deterministic models reflects the corresponding structure at the nondeterministic level. Since structures of data types are not our concern here, we make some simplifying assumptions. These simplifications are not essential for our results. They can be removed without changing the theory significantly. We assume that at the nondeterministic level all data streams carry data of the same type $C$. For our deterministic models, as foreshadowed in Section 1, we adjoin a new value $\tau$, the wait, to give a data type $D = C \cup \{\tau\}$. $D^*$ is the set of finite and denumerably infinite sequences of elements of $D$, including the empty
sequence, written as $\bot$. We shall call elements of $D^\omega$ histories. For $n \geq 1$, elements of $(D^\omega)^n$ may be called history vectors. We may write $\bot$ for any vector all of whose histories are $\bot$.

2.2. Oracle values and oracle sequences

Each deterministic model has an oracle input, which carries a stream of values which we call an oracle sequence. These sequences are conceptually distinct from data streams. For simplicity we take the values which can occur in oracle sequences to be 0 and 1. Oracle sequences are infinite sequences of 0's and 1's (Definition 2.2.1).

For our later discussion it will also be convenient to consider generalised oracle sequences, in which some values are undetermined. We shall refer to an oracle sequence as total if all of its values are 0 or 1. The oracle sequences used in the actual modelling of nondeterministic processes will be total.

We introduce the set of oracle sequences. We write $\bot$ for an undetermined element of an oracle sequence; we mean the bottom element of the flat domain $\{0, 1, \bot\}$. This bottom element should not be confused with the bottom element of $D^\omega$ introduced previously. The context should always distinguish the two.

2.2.1. Definition. An oracle sequence is a mapping $\mathbb{N}^+ \rightarrow \{0, 1, \bot\}$.

2.2.2. Definition. A total oracle sequence is a mapping $\mathbb{N}^+ \rightarrow \{0, 1\}$.

We now define fairness in a conventional way.

2.2.3. Definition. A total oracle sequence $s$ is fair if for all $n \in \mathbb{N}^+$, $\exists n', n'' > n$ such that $s(n') = 0$ and $s(n'') = 1$.

This definition extends straightforwardly to oracle sequences in general, as follows.

2.2.4. Definition. For oracle sequences $s$ and $t$ we define $s \leq_p t$ ("pointwise" partial order) to mean that for all $n$ such that $s(n) \neq \bot$, $s(n) = t(n)$.

2.2.5. Definition. An oracle sequence $s$ is fair if there is a fair total oracle sequence $t \geq_p s$. For example $\bot^\omega$ is fair.

To motivate our concept of strong fairness for oracle sequences, consider the ways in which oracle sequences are used. As illustrated in the introductory example, an oracle sequence is used in one of the two ways. It may be used for some specific computational purpose such as wait injection. A viable subtheory of our theory is obtained by assuming that wait injection is the only specific computational use of oracle sequences.
Alternatively an oracle sequence may be used for the special purpose of being subdivided, or distributed, into two oracle sequences by an oracle distributor. In this case, we want the fairness property of the original oracle sequence to be inherited by both of the subdivided oracle sequences. To achieve that, using the simple oracle distributor defined in the introductory example, we develop a suitable notion of strong fairness. First we define a simple language of positions to describe those subsequences of an oracle sequence which become oracle sequences as a result of repeated subdivisions.

2.2.6. **Definition.** We write $P$ for the set of arbitrary finite strings of $L$ and $R$. We call these strings *positions*. Precisely, $P$ can be defined inductively as follows. We write $\epsilon$ for the empty string.

1. $\epsilon \in P$.
2. If $p \in P$, then $Lp, Rp \in P$.

2.2.7. **Definition.** Next we use the notation $\text{sub}(\pi, s)$ to describe the subsequence at position $\pi$ of an oracle sequence $s$. Precisely, $\text{sub}(\pi, s)$ can be defined inductively as follows. Let $s$ be an oracle sequence and $\pi \in P$. Define the oracle sequence $\text{sub}(\pi, s)$ such that

1. If $\pi = \epsilon$, $\text{sub}(\pi, s) = s$.
2. If $\pi = L\pi_1$, $\text{sub}(\pi, s) = \text{sub}(\pi_1, \lambda(s))$.
3. If $\pi = R\pi_1$, $\text{sub}(\pi, s) = \text{sub}(\pi_1, \rho(s))$.

We also write $AS(s)$ for the set $\{\text{sub}(\pi, s) : \pi \in P\}$ of sequences at some position in an oracle sequence. We call these the *alternating* subsequences of $s$.

We give an alternative characterisation of $AS(s)$.

2.2.8. **Definition.** Given a sequence $s$ and integers $m \geq 1$, $k \geq 0$ define the sequence $s_{m, k}$ by

$$s_{m, k}^+(n) = s(m + (n - 1)2^k), \quad n \geq 1.$$ 

2.2.9. **Lemma.** The sequences $s_{m, k}^+$, $k \geq 0$, $1 \leq m \leq 2^k$ are exactly the alternating subsequences of $s$.

**Proof.** An elementary induction proves it. Details are omitted. □

To relate these subsequences to the natural number indices of the whole sequence, we shall say that an index $i$ appears at position $\pi$ if the $i$th term of the whole sequence is a term of the subsequence at $\pi$. More precisely, we make the following definitions.
2.2.10. **Definition.** For each position \( \pi \), we define integers \( m(\pi) \), \( k(\pi) \) such that

1. If \( \pi = \epsilon, m(\pi) = 1, k(\pi) = 0 \).
2. If \( \pi = L\pi', m(\pi) = m(\pi'), k(\pi) = k(\pi') + 1 \).
3. If \( \pi = R\pi', m(\pi) = m(\pi') + 2^{k(\pi')}, k(\pi) = k(\pi') + 1 \).

2.2.11. **Lemma.** For all \( \pi, s \), \( \text{sub}(\pi, s) = s_{m(\pi), k(\pi)} \).

2.2.12. **Definition.** For an index \( i \), we say \( i \) appears at position \( \pi \), if, writing \( m = m(\pi) \), \( k = k(\pi) \), \( i \) is in the set \( \{m + (n - 1)2^k | n \geq 1\} \). If \( i \) appears at \( \pi \), then for all \( s \), \( s(i) = \text{sub}(\pi, s)(n) \), where \( i = m + (n - 1)2^k \).

2.2.13. **Definition.** An oracle sequence \( s \) is **strongly fair** if every alternating subsequence of \( s \) is fair. For example \( \bot^w \) is strongly fair.

Denote the set of strongly fair total oracle sequences by \( A \). Denote the set of strongly fair partial oracle sequences by \( \Omega \).

2.2.14. **Lemma.** A sequence \( s \) is strongly fair if and only if every alternating subsequence is strongly fair.

**Proof.** See Appendix.

2.2.15. **Lemma.** Let \( s \) be strongly fair and \( s' = \text{sub}(\pi, s) \) be the alternating subsequence of \( s \) at position \( \pi \). Let \( s' \) contain only a finite number of \( \bot \) values. Then for every total \( t \) such that \( t \geq_p s \), \( \text{sub}(\pi, t) \) is fair.

**Proof.** See Appendix.

2.2.16. **Lemma.** If \( s \) is strongly fair then there is a total strongly fair \( t \geq_p s \).

**Proof.** See Appendix.

2.2.1. **Existence of total strongly fair sequences**

We have already noted that \( \bot^w \) is strongly fair. The algorithm used in the proof of Lemma 2.2.16 can then be used to construct a total strongly fair sequence.

2.2.2. **Fairness and strong fairness**

Strongly fair oracle sequences are used purely as an encoding of infinitely many fair sequences. This encoding is convenient for the purpose of continual subdivision into infinite subfamilies.

We do not claim that the use of strongly fair, instead of fair, oracle sequences is appropriate for modelling actual nondeterministic behaviour. Rather, we show in Section 6.3 that we can build a process FS which will produce arbitrary fair sequences.
from strongly fair sequences. Fair sequences are intended to be used for modelling actual nondeterministic behaviour. Thus, the use of strongly fair, instead of fair, sequences does not restrict the class of processes modelled.

2.3. A partial order for oracle sequences

The remainder of this section is not required for Sections 3 or 4, which describe the basic concepts used to build deterministic models. It is fundamental to the discussion of equivalence of deterministic models, which begins in Section 5. We begin with some notation for describing operations on oracle sequences. We have already described positions, and functions \( \lambda, \rho \) and \( \text{sub} \) for accessing subsequences at particular positions. The following function \( U \) inverts \( \lambda \) and \( \rho \).

2.3.1. Definition. Define the function \( U: \Omega \times \Omega \rightarrow \Omega \) as follows.

\[
U(s_1, s_2)(n) = \begin{cases} s_1((n+1) \text{ div } 2) & \text{if } n \text{ is odd,} \\ s_2(n \text{ div } 2) & \text{if } n \text{ is even.} \end{cases}
\]

2.3.2. Lemma. (i) \( U(\lambda s, \rho s) = s \).
(ii) \( \lambda(U(s_1, s_2)) = s_1 \).
(iii) \( \rho(U(s_1, s_2)) = s_2 \).
(iv) If \( s_1, s_2 \) are strongly fair, then \( U(s_1, s_2) \) is strongly fair.

Next we define updating of oracle sequences at arbitrary positions.

2.3.3. Definition. Let \( p \in P, s, s' \in \Omega \). Define the \( s' \) update of \( s \) at \( p \), written \([s'/p]s\), to be the result of changing to \( s' \) the subsequence of \( s \) at \( p \). Precisely, the concept can be defined inductively as follows.

1. \([s'/p]s = s'\).
2. If \( p = Lp' \), then \([s'/p]s = U([s'/p'](\lambda s), \rho s)\).
3. If \( p = Rp' \), then \([s'/p]s = U(\lambda s, [s'/p'](\rho s))\).

Next we extend the update notation to allow parallel updates at several independent positions.

2.3.4. Definition. Two positions are independent if neither is a prefix of the other. A set of positions is called independent if each pair of its elements is independent.

2.3.5. Definition. For each oracle sequence \( s \), each independent set \( \Pi \) of positions and each function \( \sigma: \Pi \rightarrow \Omega \), we define \([\sigma]s\) to be the result of modifying, for each \( \pi \in \Pi \), the subsequence of \( s \) at \( \pi \) to \( \sigma(\pi) \).

This definition succeeds because the elements of \( \Pi \) are independent, and subsequences at independent positions do not overlap.
We can now define a partial order on oracle sequences.

2.3.6. Definition. For all \( s, t \in \Omega \), \( s \leq_{AS} t \) means \( t \) is a parallel update of \( s \), say \( t = [\sigma]s \), such that for all \( \pi \in \text{dom} \sigma \), \( \sigma(\pi) \) is total and strongly fair, and \( \text{sub}(\pi, s) = \bot^\omega \).

Write \( \Omega_{AS} \) for those elements of \( \Omega \) such that \( s \geq_{AS} \bot^\omega \).

2.3.7. Lemma. \((\Omega_{AS}, \leq_{AS})\) is a partial order with least element \( \bot^\omega \) and in which increasing sequences have upper bounds (but not generally least upper bounds).

This lemma is a consequence of the following series of lemmas.

2.3.8. Lemma. The relation \( \leq_{AS} \) is reflexive and anti-symmetric.

Proof. See Appendix.

2.3.9. Lemma. The relation \( \leq_{AS} \) is transitive.

Proof. See Appendix.

2.3.10. Lemma. Every chain of elements in \((\Omega_{AS}, \leq_{AS})\) has an upper bound in \((\Omega_{AS}, \leq_{AS})\).

Proof. See Appendix.

The following observations about parallel updates will be used later in the paper; first, a partial order on parallel updates with the same domain.

2.3.11. Definition. For all parallel updates \( \sigma, \sigma' \) with the same domain \( \Pi \), we write \( \sigma \leq_{AS} \sigma' \) to denote that for all \( \pi \in \Pi \), \( \sigma(\pi) \leq_{AS} \sigma'(\pi) \).

2.3.12. Lemma. If \( \sigma, \sigma' \) are parallel updates with the same domain \( \Pi \), if \( \delta, \delta' \in \Omega \) and if \( \sigma \leq_{AS} \sigma' \) and \( \delta \leq_{AS} \delta' \), then \([\sigma] \delta \leq_{AS} [\sigma'] \delta' \).

Proof. See Appendix.

2.3.13. Lemma. Let \( \delta \) be total and strongly fair, and let \( \sigma \) be an update such that \( \text{dom} \sigma \) is finite and for all \( \pi \in \text{dom} \sigma \), \( \sigma(\pi) = \bot^\omega \). Then \([\sigma] \delta \geq_{AS} \bot^\omega \).

Proof. See Appendix.

We note here for later use the following operation of concatenation on independent sets.
2.3.14. Definition. For all independent sets of positions $\Pi_1$ and $\Pi_2$, define \( \text{concat}(\Pi_1, \Pi_2) \) to be the set of componentwise concatenations of $\Pi_1$ and $\Pi_2$, i.e.
\[
\text{concat}(\Pi_1, \Pi_2) = \{ \pi_1\pi_2 : \pi_1 \in \Pi_1, \pi_2 \in \Pi_2 \}.
\]

2.3.15. Lemma. For all independent sets $\Pi_1$ and $\Pi_2$, $\text{concat}(\Pi_1, \Pi_2)$ is an independent set.

3. Deterministic processes used for modelling

3.1. Introduction

Following [S], we characterise deterministic processes as continuous functions of some type \((D^\infty)^m \times A \to (D^\infty)^n\), $m, n \geq 0$. A function of this type characterises a deterministic data flow process with $m$ input data ports, a single oracle input port and $n$ output ports. Note that in oracle inputs, modelling functions are continuous with respect to the standard prefix ordering on oracle sequences (regarded simply as sequences of 0's and 1's). The splitting process $\text{Dist}$, already defined, is clearly continuous with respect to the standard partial order.

For brevity, we write $[m\to n]$ for $(D^\infty)^m \times A \to (D^\infty)^n$. For $h \in (D^\infty)^m$, $\delta \in A$ and $f \in [m\to n]$ we may write $f(h)$ instead of $f(h, \delta)$ when the abbreviation is not ambiguous in context. We may also write $\perp$ for the function with constant value $\perp$.

For reasons mentioned in Section 1, we shall restrict the class of deterministic processes to be used in our models. The intention is not to restrict the class of nondeterministic processes which can be modelled, but rather to fix a convenient structure for the deterministic models. The deterministic processes used in our deterministic modelling satisfy a modelling condition. Intuitively, the modelling condition can be motivated by interpreting a wait data item as a unit time delay. More specifically, a wait may be regarded as terminating a transmission at a given time. With this interpretation, we may speak of data items occurring between the $(n-1)$th and the $n$th wait as occurring at time $n-1$. This interpretation is not necessary to the analysis but is an acceptable, elementary, operational interpretation. The modelling condition then imposes on deterministic processes requirements which may be interpreted as follows.

1. Output at time zero is totally defined before input is read (the initialising condition).

2. Input affects only later output (the causality condition). More precisely, for each $n$, output at times less than $n$ is independent of input at times greater than or equal to $n$.

No further restrictions are imposed on the ways in which processes may respond to the timing patterns of inputs. In the following technical development, we allow infinite sequences with finitely many waits. Such sequences may be excluded from the
analysis, if so desired, on operational grounds; the analysis is unaffected. Since the
operational semantics for the Kahn semantics of deterministic data flow is well
known, the given operational meaning for waits and the interpretation of oracles as
sources of nondeterminism fixes a definite operational interpretation of the processes
we consider. Technically, the modelling condition ensures that the data streams
internal to a network of modelling processes are always sufficiently padded with waits
to be infinite and to avoid spurious deadlocks.

3.2. Domination

Our definition of modelling is based on the following domination relation on
history vectors. Intuitively, $h$ dominates $k$ can be thought of as meaning that $h$ extends
$k$ and, in the case that $k$ is finite, the maximum time for which $h$ is totally defined is
greater than the maximum time for which $k$ is totally defined.

3.2.1. Definition. For history vectors $h = (h_1, \ldots, h_m)$, $k = (k_1, \ldots, k_m)$, we say $h$ dominates $k$, written as $h \triangleright k$, if for all $i$, $1 \leq i \leq m$, either $h_i = k_i$ and $h_i, k_i$ are infinite or $h_i > k_i$ and $h_i$ contains strictly more waits than $k_i$. For example, $\top \triangleright \bot$, but not $\bot \triangleright \top$.

3.2.2. Lemma. (i) $h \triangleright h$ if all components of $h$ are infinite.
(ii) $h \triangleright k$ and $k \triangleright h$ implies $h = k$.
(iii) $h \triangleright k$ and $k \triangleright k'$ implies $h \triangleright k'$.
(iv) If $(h_n)$ is an increasing sequence and for all $n$, $h_n \triangleright k_n$, then $\operatorname{lub}_n h_n \triangleright k_n$.
(v) If $(k_n)$ is an increasing sequence and for all $n$, $h \triangleright k_n$, then $h \triangleright \operatorname{lub}_n k_n$.
(vi) $h \triangleright k$ and $k \triangleright k'$ implies $h \triangleright k'$.
(viii) $h \triangleright k$ and $k \triangleright k'$ implies $h \triangleright k'$.

The following extension of domination to continuous functions is a key concept for
our analysis, and also leads directly to the causality condition.

3.2.3. Definition. Let $f, g \in [r \rightarrow s]$. We say $f$ dominates $g$ if for all $h \triangleright k$, $\delta \in \Delta, f(h, \delta) \triangleright g(k, \delta)$. We write $f \triangleright g$.

3.2.4. Definition. Let $f \in [r \rightarrow s]$. We say $f$ is causal if for all $h \triangleright k$, $\delta \in \Delta, f(h, \delta) \triangleright f(k, \delta)$. Note that this is equivalent to $f \triangleright f$.

3.2.5. Lemma. If $f$ is causal and all components of $h$ are infinite then all components of $f(h, \delta)$ are infinite.

Proof. Since $h$ is infinite, $h \triangleright h$. Since $f$ is causal, $f(h) \triangleright f(h)$ and therefore $f(h)$ is
infinite. $\Box$

We extend the above definitions to vectors of functions as follows.
3.2.6. **Definition.** Let \( f = (f_1, \ldots, f_n) \), \( g = (g_1, \ldots, g_n) \) be \( n \)-tuples of functions. We say \( f \) dominates \( g \) if \( f_i d g_i \) for all \( i, 1 \leq i \leq n \). We write \( f d g \). We say \( f \) is causal if \( f d f \).

The initialising condition which completes the concept of modelling is also easily expressed in terms of domination.

3.2.7. **Definition.** We say \( f \in [r\rightarrow s] \) is initialising if \( f(\bot, \bot) d \bot \). A vector \( f \) of functions is initialising if each component of \( f \) is initialising.

3.2.8. **Definition.** A process \( f \) is a modelling process if it is initialising and causal.

**Examples**

We present here some elementary examples of modelling and nonmodelling processes.

**Example (1):** For an arbitrary finite sequence \( a \) and an arbitrary sequence \( b \), the process \( f \) defined by

\[
f(X, \delta) = a \cdot \tau \cdot b \quad \text{for all} \; X, \delta
\]

is initialising, since clearly \( f(\bot, \bot) d \bot \).

**Example (2):** The identity process \( \text{Id} \) defined by

\[
\text{Id}(X, \delta) = X \quad \text{for all} \; X, \delta
\]

is noninitialising. To see that, note that \( \text{Id}(\bot, \bot) = \bot \) and \( \bot \) does not dominate \( \bot \). The identity process is causal since if \( h d k \), then for all \( \delta \), \( \text{Id}(h, \delta) d \text{Id}(k, \delta) \).

**Example (3):** The process \( f \) defined below is initialising but noncausal.

\[
f(X, \delta) = \tau \cdot f'(X, \delta), \quad f'(\bot, \delta) = \bot.
\]

To see that \( f \) is noncausal, note that \( \tau \) dominates \( \bot \), but \( f(\tau) = f(\bot) = \tau \) and \( \tau \) does not dominate \( \tau \).

We now give some examples of modelling processes.

**Example (4).** A modelling identity process: \( \text{MId}(X, \delta) = \tau \cdot X \).

**Example (5).** A modelling cons process: The process cons has two nonoracle inputs \( X \) and \( Y \). Intuitively, the process waits for a nonwait input on \( X \). When such a data item appears, it is output and then all data items appearing on input \( Y \) are output.

\[
\text{cons}(X, Y, \delta) = \tau \cdot \text{cons}'(X, Y, \delta), \quad \text{cons}'(\bot, Y, \delta) = \bot,
\]

\[
\text{cons}'(\tau \cdot X, Y, \delta) = \tau \cdot \text{cons}'(X, Y, \delta),
\]

\[
\text{cons}'(a \cdot X, Y, \delta) = a \cdot Y, \quad a \neq \tau.
\]

**Example (6).** A nondeterministic distributor: The distributor process has one input and two outputs. The process distributes a wait in its input stream to both outputs.
When a nonwait data item is encountered, the process consults its oracle input stream. An oracle value of 0 causes the data item to be distributed to the left output and an oracle value of 1 causes the data item to be distributed to the right. We describe the behaviours of the distributor by defining two modelling processes \( \text{dist}_1 \) and \( \text{dist}_2 \), defining the left- and right-output streams, respectively.

\[
\begin{align*}
\text{dist}_1(X, \delta) &= \tau \cdot \text{dist}_1'(X, \delta), \\
\text{dist}_1'(\bot, \delta) &= 1, \\
\text{dist}_1'(\tau, X, \delta) &= \tau \cdot \text{dist}_1'(X, \delta), \\
\text{dist}_1'(a.X, 0, \delta) &= a \cdot \text{dist}_1'(X, \delta), \ a \neq \tau, \\
\text{dist}_1'(a.X, 1, \delta) &= \tau \cdot \text{dist}_1'(X, \delta), \ a \neq \tau. \\
\end{align*}
\[
\begin{align*}
\text{dist}_2(X, \delta) &= \tau \cdot \text{dist}_2'(X, \delta), \\
\text{dist}_2'(\bot, \delta) &= 1, \\
\text{dist}_2'(\tau, X, \delta) &= \tau \cdot \text{dist}_2'(X, \delta), \\
\text{dist}_2'(a.X, 0, \delta) &= \tau \cdot \text{dist}_2'(X, \delta), \ a \neq \tau, \\
\text{dist}_2'(a.X, 1, \delta) &= a \cdot \text{dist}_2'(X, \delta), \ a \neq \tau.
\end{align*}
\]

The choices made by the distributor process are determined by the strongly fair oracle input. In applications, we may wish to use fair or even unfair sequences of 0's and 1's to drive distribution processes. This may be achieved by using the processes FS and US, defined in Section 6.3, which produce, respectively, arbitrary fair and unfair sequences. Then processes analogous to \( \text{dist} \) can be defined in which choices are determined by a data input connected to FS or US.

The modelling functions \((D^\infty)^m \rightarrow (D^\infty)^m\) form a complete subset of the larger cpo of continuous functions. However, it is not appropriate to focus attention purely on this subset, intuitively because the modelling functions represent limiting behaviours which are approximated by nonmodelling processes. Technically, this is reflected in such facts as those stating that there is no least modelling function.

As outlined in the next section, we restrict the class of operators modelling network construction schemes so that the least fixed points of such operators are modelling functions. Those least fixed points may be calculated as least upper bounds of, in general, nonmodelling functions.

4. Operators used to model networks

4.1. Introduction

A fundamental requirement for the discussion of nondeterministic processes is to have a vocabulary for describing the contexts into which a process may be placed. Intuitively, such a context is a network of processes which has a “hole” into which the process under discussion may be placed. We formalise this requirement by considering (continuous) operators on the continuous functions which model deterministic processes. Intuitively, when a process \( f \) is placed in the “hole” in a context \( F \), the resulting network is \( F(f) \). More generally, operators can also be used to define functions recursively, (see Section 4.2). It is convenient for our analysis to allow contexts with several “holes”. That in turn makes it natural to consider operators for
which the values as well as the arguments are vectors of functions. Accordingly we begin as follows.

4.1.1. **Definition.** An operator is a continuous function from \( m \)-tuples to \( n \)-tuples of functions. A typical operator will be a continuous function from \([p_1 \rightarrow q_1] \times \cdots \times [p_m \rightarrow q_m]\) to \([r_1 \rightarrow s_1] \times \cdots \times [r_n \rightarrow s_n]\). Intuitively, this operator represents an \( n \)-tuple of contexts. Each of the contexts has \( m \) "holes". For each context, the \( i \)th "hole" can be filled by a deterministic process of type \([p_i \rightarrow q_i]\). For the \( j \)th context, when all "holes" are filled the resulting network defines a deterministic process of type \([r_j \rightarrow s_j]\), \( j = 1, \ldots, n \).

4.1.2. **Definition.** For operators \( F \) and \( G \) of compatible types we denote the composite operator \( FG \). That is, for all function vectors \( f \) in the domain of \( G \), \((FG)(f) = F(G(f))\).

The modelling condition on functions extends naturally to operators as follows.

4.1.3. **Definition.** An operator \( F \) is dominance preserving if whenever \( f \triangleright q \) then also \( F(f) \triangleright F(q) \).

4.1.4. **Lemma.** If \( F, G \) are dominance preserving then so is \( FG \).

**Proof.** Evident. \( \square \)

4.1.5. **Definition.** An operator \( F \) is initialising if \( F(\bot) \) is initialising.

4.1.6. **Lemma.** If \( F \) is initialising then \( FG \) is initialising.

**Proof.** \( FG(\bot) \geq F(\bot) \) by monotonicity of \( F \). Also \( F(\bot)(h) \triangleright \bot \) for all \( h \). Thus \( FG(\bot)(h) \triangleright \bot \) for all \( h \). \( \square \)

4.1.7. **Lemma.** If \( F \) is initialising, then \( F(\bot) \triangleright \bot \).

4.1.8. **Definition.** An operator is a modelling operator if it is initialising and dominance preserving.

It should be noted however that we shall actually make use of only a specific class \( \text{MO} \) of modelling operators, to be defined in Section 4.3.

4.1.9. **Corollary.** If \( F \) and \( G \) are modelling operators and are composable then \( FG \) is also a modelling operator.

4.1.10. **Lemma.** If \( F \) is a modelling operator and \( f \) is a modelling function in the domain of \( F \), then \( F(f) \) is also a modelling function.
Proof. We have to show that $F(f)$ is initialising and causal, given that $f$ has these properties.

(i) Initialising: For all $h$, $F(f)(h) \geq F(\bot)(h)$ by monotonicity of $F$, since $F$ is initialising.

(ii) Causal: $F(f) \mathbf{d} F(f)$ is immediate since $f$ is causal and $F$ is dominance preserving.

4.2. Recursive definitions of modelling processes

The continuous operators discussed in Section 4.1 can be used more generally for recursive definitions of deterministic processes. This is the core of the Kahn semantics [5]. Since we impose a restriction on modelling processes, we show that recursive definitions using modelling operators define modelling processes. This shows that the modelling processes are closed under network construction using modelling operators. It also shows that the class of modelling processes is closed under recursion using modelling network schemes. In the next section we exhibit a useful class of modelling network construction operators.

4.2.1. Theorem. If $F$ is a modelling operator on $[r_1 \rightarrow s_1] \times \cdots \times [r_n \rightarrow s_n]$ and $f$ is the least fixed point of $f = F(f)$, then $f$ is modelling.

Proof. $f = \text{lub } F'(\bot)$. Since $F(\bot)$ is initialising and $f \geq F(\bot)$, $f$ is initialising. To show causality, we consider each component $f_j$ of $f = (f_1, \ldots, f_n)$ and show that for all $h \mathbf{d} k$, $f_j(h) \mathbf{d} f_j(k)$. If $f_j(k)$ is infinite, i.e. all components of $f_j(k)$ are infinite, then since $h \geq k$ and $f_j$ is monotonic, $f_j(h)$ is infinite and thus $f_j(h) \mathbf{d} f_j(k)$. We write $f_j(k)_i$ for the $i$th component of the output of $f_j(k)$.

Now $f(k) = F^n(\bot)(k) = (\text{lub}_q F^q(\bot))(k) = \text{lub}_q (F^q(\bot))(k)$ and so, for each finite component of $f_j(k)$, say $f_j(k)_i$, $f_j(k)_i = (F^q(\bot))_j(k)_i$ for some $q$. Now

$$f_j(h)_i \geq ((F^{q+1}(\bot))_j h)_i = ((F^q(F(\bot)))_j h)_i \mathbf{d} ((F^q(\bot))_j k)_i \text{ from Lemmas 4.1.7 and 4.1.10}$$

$$= f_j(k)_i.$$

4.3. The class MO of modelling operators

The class MO of operators considered here is the class of operators to be used later in the paper for modelling network constructions. First we describe the basic classes of operators to be used in the definition of MO. Note that most of these basic operators are not themselves in MO.

From this point we make occasional use of $\lambda$ notation in the description of functions. It should not be confused with the $\lambda$ operation on oracle sequences defined in Section 1.
4.3.1. Definition (modelling constant operators). Let \( g \) be a modelling function of type \([r \to s]\) and let \( f_i \) be variables of type \([p_i \to q_i]\), \( i = 1, \ldots, m \). Then \( \lambda(f_1, \ldots, f_m).g \) is a modelling constant operator from \([p_1 \to q_1] \times \cdots \times [p_m \to q_m]\) to \([r \to s]\), whose value is \( g \) everywhere.

4.3.2. Definition (causal constant operators). Let \( c \) be a causal function of type \([r \to s]\) and let \( f_i \) be variables of type \([p_i \to q_i]\), \( i = 1, \ldots, m \). Then \( \lambda(f_1, \ldots, f_m).c \) is a causal constant operator from \([p_1 \to q_1] \times \cdots \times [p_m \to q_m]\) to \([r \to s]\), whose value is \( c \) everywhere.

4.3.3. Definition (projection operators). Let \( f_i \) be of type \([p_i \to q_i]\), \( i = 1, \ldots, m \). Then \( \lambda(f_1, \ldots, f_m).f_i \) is a projection operator from \([p_1 \to q_1] \times \cdots \times [p_m \to q_m]\) to \([p_i \to q_i]\). We shall denote this operator by \( P_i \), the types being implied by the context.

4.3.4. Definition (identity operator). We denote the identity operator by \( \text{Id} \). It is defined by \( \text{Id}(f) = f \).

4.3.5. Definition (disjoint union operator). Disjoint union models the operation of forming a process from two component processes by grouping them together without interconnection. In our theory this operation is not naive because each of the component processes requires an oracle sequence. These two oracle sequences are obtained from the single oracle sequence of the disjoint union by an oracle distribution process \( \text{Dist} \) as defined in Section 1. Thus, the disjoint union operation is as depicted in Fig. 2. The input streams \( \text{OS} \) are the oracle streams for each of the processes.

For each pair \([p \to q]\) and \([r \to s]\) of function types there is a disjoint union operator \( \text{DU} \) from \([p \times q] \times [r \to s]\) to \([[(p + r) \to (q + s)]\), defined by

\[
(\text{DU}(f_1, f_2))(h_1, \ldots, h_p, h_{p+1}, \ldots, h_{p+r}, \delta) = (f_1(h_1, \ldots, h_p, \lambda(\delta)), f_2(h_{p+1}, \ldots, h_{p+r}, \rho(\delta)).
\]

Fig. 2. Sketch of the disjoint union operation.
4.3.6. Definition (composition operator). The composition operators form a process from two component processes by feeding the output of one into the input of the other. As in the case of disjoint union, the component processes receive independent oracle inputs as a result of splitting the single oracle sequence of the composition.

Let \( g \) be of type \([p+q]\) and \( f \) of type \([q-r]\). Then the composition operator \( \text{Comp} \) from \([q\rightarrow r] \times [p\rightarrow q]\) to \([p\rightarrow r]\) is defined by

\[
\text{Comp}(f, g)(h, \delta) = f(g(h, \lambda(\delta)), \rho(\delta)).
\]

Figure 3 illustrates the composition operator.

4.3.7. Definition (link operator). These operators are designed to form a new process from a given process by connecting an input port to an output port. Note that linking is more general than composition since it provides for the output of a process to be fed back to its own input. In this case we need to calculate the history on the looped data stream as a least fixed point, following the Kahn semantics. Linking and disjoint union together are sufficient to allow the construction of arbitrary finite networks.

Let \( f \in [p\rightarrow q] \), \( i \in \{1, \ldots, p\} \) and \( k \in \{1, \ldots, q\} \). Intuitively \( \text{LINK}_k(f) \in [(p-1)\rightarrow (q-1)] \) is as depicted in Fig. 4. Formally, it is defined as follows. For \( r=1, \ldots, k-1, k+1, \ldots, q \)

\[
\text{LINK}_k(f)(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p, \delta)_r = [f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)_r].
\]
where $H$ is the least fixed point of

$$H = \left[ f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta) \right]_k.$$ 

Note that oracle lines cannot be linked.

**4.3.8. Definition (function tupling operators).** For $i = 1, \ldots, m$ let $O_i$ be an operator from $[p_1 \rightarrow q_1] \times \cdots \times [p_n \rightarrow q_n]$ to $[r_i \rightarrow s_i]$. Define the operator $[O_1, \ldots, O_m]$ from $[p_1 \rightarrow q_1] \times \cdots \times [p_n \rightarrow q_n]$ to $[r_1 \rightarrow s_1] \times \cdots \times [r_m \rightarrow s_m]$ by

$$[O_1, \ldots, O_m](f_1, \ldots, f_n) = (O_1(f_1, \ldots, f_n), \ldots, O_m(f_1, \ldots, f_n)).$$

This completes the definition of the basic classes of operators used for the definition of MO. Next we define a set DP of dominance-preserving operators.

**4.3.9. Definition.** The set DP is the least set of operators closed under the following conditions.

1. The modelling constant operators, the causal constant operators and the projection operators belong to DP.
2. $DU, Id, Comp \in DP$.
3. For all $G_1, \ldots, G_m \in DP$, $[G_1, \ldots, G_m] \in DP$.
4. For all $G_1, G_2 \in DP$, $G_1 G_2 \in DP$.

It is straightforward to verify that all operators in DP are dominance preserving. Now we define the class MO recursively as follows.

**4.3.10. Definition.** The set MO is the least set of operators closed under the following conditions.

1. The modelling constant operators are in MO.
2. For all $G \in MO$ and $D \in DP$, $Comp[G, D] \in MO$.
3. For all $G \in MO$, $Link[G] \in MO$.
4. For all $G \in MO$ and every causal constant operator $C$, $Comp[C, G] \in MO$.
5. For all $G_1, G_2 \in MO$, $DU[G_1, G_2] \in MO$.
6. For all $G_1, \ldots, G_m \in MO$ and all $i$ such that $1 \leq i \leq m$, $P_i[G_1, \ldots, G_m] \in MO$, where $P_i$ is the $i$th projection operator (Definition 4.3.2).
7. For all $G_1, \ldots, G_m \in MO$, $[G_1, \ldots, G_m] \in MO$.
8. For all $G \in MO$ and $D \in DP$, $GD \in MO$.
9. For all $G_1, G_2 \in MO$, $G_1 G_2 \in MO$.

**Remark.** In the next section we show that all operators in MO are modelling. It follows from Theorem 4.2.1 that for each operator $G$ in MO the least fixed point of $G$ is a modelling process. That least fixed point defines a modelling constant operator which is also in MO.
4.4. All MO operators are modelling

Most cases are easy to check. We present here proofs for cases (2) and (3) under Definition 4.3.10.

4.4.1. Lemma. If $G$ is modelling and $D$ is dominance preserving, then $\text{Comp}[G,D]$ is modelling.

Proof. Firstly, $\text{Comp}[G,D]$ is initialising since

$$\text{Comp}[G,D](f)(\bot, \delta) = G(f)(D(f)(\bot, \lambda(\delta)), \rho(\delta)) \geq G(f)(\bot, \rho(\delta)) \downarrow.$$  

To show that $\text{Comp}[G,D]$ is dominance preserving, assume $h \downarrow k$ and $f \downarrow g$. Then, since $D$ is dominance preserving, $D(f)(h, \lambda(\delta)) \downarrow D(g)(k, \lambda(\delta))$ and since $G$ is dominance preserving, $G(f)(D(f)(h, \lambda(\delta)), \rho(\delta)) \downarrow G(g)(D(g)(k, \lambda(\delta)), \rho(\delta)).$ $\square$

4.4.2. Lemma. $\text{LINK}_i^G$ is initialising whenever $G$ is initialising.

Proof. We have

$$(\text{LINK}_i^G)(\bot) = \text{LINK}_i(G(\bot))$$

$$\text{LINK}_i(G(\bot))(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p, \delta)$$

$$= G(\bot)((x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)$$

$$\downarrow_\bot$$ since $G$ is initialising. $\square$

4.4.3. Lemma. $\text{LINK}_i^G$ is dominance preserving provided $G$ is initialising and dominance preserving.

Proof. Let $f \downarrow g$ and $h \downarrow k$. Then $f(h, \delta) \downarrow g(k, \delta)$. It suffices to show that

if $(h_1, \ldots, h_{i-1}, h_{i+1}, \ldots, h_p) \downarrow (k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_p)$ then $H \downarrow H'$,

where $H$ is the least fixed point of $H = [G(f)(h_1, \ldots, h_{i-1}, H, h_{i+1}, \ldots, h_p, \delta)]_k$ and $H'$ is the least fixed point of $H' = [G(g)(k_1, \ldots, k_{i-1}, H', k_{i+1}, \ldots, k_p, \delta)]_k$.

Consider the approximations to $H, H'$:

$$H_0 = [G(f)(h_1, \ldots, h_{i-1}, \bot, h_{i+1}, \ldots, h_p, \delta)]_k,$$

$$H_i = [G(f)(h_1, \ldots, h_{i-1}, H_{i-1}, h_{i+1}, \ldots, h_p, \delta)]_k,$$

$$H_0 = [G(g)(k_1, \ldots, k_{i-1}, \bot, k_{i+1}, \ldots, k_p, \delta)]_k,$$

$$H_i = [G(g)(k_1, \ldots, k_{i-1}, H'_{i-1}, k_{i+1}, \ldots, k_p, \delta)]_k.$$

We show below that $H_0 \downarrow H'_0$, and by induction that $H_{i+1} \downarrow H'_i$ for all $0 \leq j \leq i$. Then, since $H \geq H_i$ for all $i$, $H \downarrow H'_i$ for all $i$. Thus, $H \downarrow H'$, by Lemma 3.2.2(v).
First we show that $H_1 \text{d} H_0$. It is clear that $H_0 \text{d} \perp$ since $G(f)$ is initialising. Therefore,

$$H_1 = [G(f)(h_1, \ldots, h_{i-1}, H_0, h_{i+1}, \ldots, h_p, \delta)]_k$$

$$\text{d}[G(g)(k_1, \ldots, k_{i-1}, \perp, k_{i+1}, \ldots, k_p, \delta)]_k = H'_0$$

since $f \text{d} g, G$ is dominance preserving and $h \text{d} k$.

Now assume inductively that $H_{i+1} \text{d} H'_j$ for all $0 \leq j \leq i$. We show that $H_{i+2} \text{d} H'_j$ for all $0 \leq j \leq i+1$. First, $H_{i+2} \geq H_{i+1}$ and therefore, $H_{i+2} \text{d} H'_j$ for all $0 \leq j \leq i$, from Lemma 3.2.2(vi). It remains to show that $H_{i+2} \text{d} H'_{i+1}$. For this note that

$$H_{i+2} = [G(f)(h_1, \ldots, h_{i-1}, H_{i+1}, h_{i+1}, \ldots, h_p, \delta)]_k$$

$$\text{d}[G(g)(k_1, \ldots, k_{i-1}, H'_{i+1}, k_{i+1}, \ldots, k_p, \delta)]_k \text{ by inductive assumption}$$

$$= H'_{i+1}.$$

4.5. Simultaneous and iterated linking

So far we have discussed linking a single output to a single input. The extension of our definitions and results to the case of the simultaneous linking of a number of ports will follow from the result that step-by-step linking is equivalent to parallel linking. This result is known [2, 15] but is not readily accessible in this form.

4.5.1. Theorem (associativity of deterministic linking). An arbitrary number of parallel links is equivalent to a series of step-by-step linkings.

Proof. See Appendix.

5. Equivalence classes of modelling processes

In this section we define and analyse a notion of behaviour equivalence on modelling processes. It is intended to capture the idea that two equivalent models have no difference which is observable at the nondeterministic level, either by direct observation or by indirect means of observing the behaviours of the two processes in some arbitrary context. We also show in Theorem 6.4.1 that it is the broadest equivalence relation with this property: two inequivalent models can be observed, with the aid of a suitable context, to be inequivalent.

5.1. Behaviour equivalence

It is convenient to define and analyse behaviour equivalence by means of the following subequivalence.
5.1.1. Definition. Let $f, g$ be modelling processes of type $[r \rightarrow s]$. We say $f$ is behaviour subequivalent to $g$ written as $f \preceq g$ if for all infinite $X$ and $\delta \in \Delta$ there exists a $\delta' \in \Delta$ such that $f(X, \delta) = g(X, \delta')$. More generally, if $f = (f_1, \ldots, f_m)$, $g = (g_1, \ldots, g_m)$ are vectors of modelling processes of the same type $f \preceq g$ means that $f_i \preceq g_i$, $i = 1, \ldots, m$.

5.1.2. Lemma. Behaviour subequivalence is reflexive and transitive.

Proof. Immediate. \( \square \)

5.1.3. Definition. We say modelling processes $f$ and $g$ are behaviour equivalent written as $f \equiv g$ iff $f \preceq g$ and $g \preceq f$; and similarly, for vectors $f$ and $g$.

5.1.4. Lemma. Behaviour equivalence is an equivalence relation.

Proof. Immediate from Lemma 5.1.2. \( \square \)

5.1.5. Lemma. (associativity of disjoint union with respect to behaviour equivalence).

\[
\text{DU}(\text{DU}(f_1, f_2), f_3) \preceq \text{DU}(f_1, \text{DU}(f_2, f_3)).
\]

Proof. We show that

\[
\text{DU}(\text{DU}(f_1, f_2), f_3) \preceq \text{DU}(f_1, \text{DU}(f_2, f_3)).
\]

The other subequivalence is shown symmetrically.

Consider the diagrams in Fig. 5. The diagram on the left in Fig. 5 labels the strongly fair oracles arriving at $f_1, f_2, f_3$ as $\delta_1, \delta_2, \delta_3$. It clearly suffices to take $\delta'$ as $U(\delta_1, U(\delta_2, \delta_3))$. \( \square \)

5.2. Preservation of behaviour equivalence under network construction: introduction

Our aim in this section is to prove a theorem whose intuitive content is: if two processes are behaviour equivalent then so are their substitutions in any modelling context. To achieve this result we must first give a formal definition of the modelling contexts being considered. Recall that we construct network processes only as the least fixed points $z$ of equations

\[
z = G(z),
\]

where $G$ is an operator in $\text{MO}$. Thus, each result of substituting a process into a modelling context is such a $z$. Our formal model for substituting a process $f$ to achieve $z = G(z)$ is

\[
G(w) = F(f, w),
\]

where $F \in \text{MO}$ also. Thus, a formal statement of the theorem to be proved is as follows.
5.2.1. Theorem. For all \( F \in \text{MO} \) of type \( t \times u \rightarrow u \) and all modelling processes \( f \) and \( g \) of type \( t \), if \( f \preceq g \) and if \( y, z \) are the least fixed points of \( y = F(f, y), z = F(g, z) \), respectively, then \( y \preceq z \).

For convenience and generality, we extend this result to vectors \( f, g \) of functions, and to operators with vector results. Thus, the theorem to be proved is the following. Here the notation \( F(f, @) \), for example, indicates some partition of the arguments of \( F \) into two vectors.

5.2.2. Theorem. For all \( F \in \text{MO} \) of type \( t \times u \rightarrow u \) and all \( f \) and \( g \) of type \( t \), if \( f \preceq g \) and if \( y, z \) are the least fixed points of \( y = F(f, y), z = F(g, z) \), respectively, then \( y \preceq z \).

It is also convenient, and sufficient, to prove the corresponding result for behaviour subequivalence. Thus, we actually prove the following.

5.2.3. Lemma. For all \( F \in \text{MO} \) of type \( t \times u \rightarrow u \) and all \( f \) and \( g \) of type \( t \), if \( f \preceq g \) and if \( y, z \) are the least fixed points of \( y = F(f, y), z = F(g, z) \), respectively, then \( y \preceq z \).

We now consider the concepts which are needed for our proof of this result.

5.3. Compensating for subequivalence: introduction

Our argument for Lemma 5.2.3 can be sketched intuitively as follows. To simplify the sketch we assume, as in Theorem 5.2.1, that the two arguments for \( F \) are single functions rather than vectors of functions.
(i) For each input $X, \delta$ to $f$ a change of $f$ to $g$ can be compensated by changing $\delta$ to some suitable $\delta'$, without changing $X$.

(ii) Since $F \in \text{MO}$, the network described by $z = F(f, z)$ depends on $f$ in a simple way. Instances of $f$ are merely “plugged into holes” in the network.

(iii) The network supplies each instance of $f$ with an oracle sequence in a simple way. Writing $s$ for the oracle sequence supplied to the whole network, each instance of $f$ is supplied with an oracle sequence $\text{sub}(\pi, s)$ for some position $\pi$. The several instances of $f$ are supplied with nonoverlapping subsequences of $s$.

(iv) Thus for each input to the network, a change from $f$ to $g$ can be compensated by making suitable changes to those nonoverlapping subsequences of $s$. Overall, that amounts only to a change in the network’s oracle sequence $s$. Accordingly, the process defined by using $f$ is subequivalent to the process defined by using $g$.

The idea just sketched exploits special properties of the operator $F$. To make the argument precise we introduce notation as follows. First, it is convenient to weaken the concept of subequivalence.

5.3.1. Definition. We say that $\delta'$ compensates $f$ to $g$ on $X, \delta$ if $f(X, \delta) \geq g(X, \delta')$.

To introduce our notation, consider the example of an operator $F$ from $\text{MO}$ of some type $[p_1 \to q_1] \times \cdots \times [p_m \to q_m] \to [r \to s]$ and an arbitrary argument $f$ of $F$, of type $[p_1 \to q_1] \times \cdots \times [p_m \to q_m]$. We write $F(f)$ for the value of $F$ at $f$. In this introduction we make various assertions about $F$ without proof, for the sake of motivation. Proofs, as required, are given later. All of the following points are significant for our argument.

(a) Each component in a network can be uniquely identified by a path from the network oracle input, through oracle distributors, to that component. We already have (Definition 2.2.6) a notation for describing these paths. In particular, there is some independent set $\Pi$ of positions which are the positions in the network at which instances of the arguments of $F$ are substituted. For example, if $F = \text{DU}$, one instance of each argument is substituted at positions $L$ and $R$; so, in this case the set $\Pi$ is $\{L, R\}$.

(b) A function $w: \Pi \to \{1, \ldots, m\}$ can describe, for each position $\pi \in \Pi$, the ordinal of the argument substituted at $\pi$. For the $\text{DU}$ example given above,

$$w(L) = 1, \quad w(R) = 2.$$  

(c) For each argument $f$ of $F$, each input $(X, \delta)$ to $F(f)$ and each position $\pi \in \Pi$, the process at $\pi$ in $F(f)$ has some input data stream which we write $I(f, \pi, X, \delta)$. Note that its oracle input is $\text{sub}(\pi, \delta)$.

We are interested in the following properties of these structures. Again, this introductory discussion has no proofs.

(d) In the notation of (c), the type of the data stream $I(f, \pi, X, \delta)$ is $p_{w(\pi)}$.

(e) The function $I$ introduced in (c) is monotonic in its first and third arguments.

(f) The last and the most important property we consider is, intuitively, as follows.
Whenever changes to an argument of \( F \) can be compensated locally, they can be compensated globally. In the notation of (c), and writing \( f = (f_1, \ldots, f_n) \), the property is as follows.

For all functions \( \sigma: \Pi \to A \), if for all \( \pi \in \Pi \), writing \( k = w(\pi), \sigma(\pi) \) compensates \( f_k \) to \( g_k \) on \( I(f, \pi, X, \delta), \text{sub}(\pi, \delta) \) then \( [\sigma] \delta \) compensates \( F(f) \) to \( F(g) \) on \( X, \delta \).

To reason efficiently about such situations we give an abstract formulation of it as follows. First we deal with the special case discussed above.

5.3.2. Definition. For every operator \( F \) from \( \text{dom} F = \prod [p_1 \rightarrow q_1] \times \cdots \times [p_m \rightarrow q_m] \) to \( [r \rightarrow s] \) a compensation structure for \( F \) is a triple \( (\Pi, w, \text{I}) \) with the following properties.

(a) \( \Pi \) is a finite independent set of positions.
(b) \( w \) is a function from \( \Pi \) to \( \{1, \ldots, m\} \).
(c) \( \text{I} \) is a function from \( \text{dom} I = \text{dom} F \times \Pi \times r \times A \) to \( \bigcup_{i=1}^{m} p_i \).
(d) For all \( (f, \pi, X, \delta) \in \text{dom} I, I(f, \pi, X, \delta) \in p_{w(\pi)} \).
(e) \( I \) is monotonic in its first and third arguments.
(f) For all updates \( \sigma : \Pi \to A \), if for all \( \pi \in \Pi, \sigma(\pi) \) compensates \( f_{w(\pi)} \) to \( g_{w(\pi)} \) on \( I(\pi, X, \delta), \text{sub}(\pi, \delta) \), then \( [\sigma] \delta \) compensates \( F(f) \) to \( F(g) \) on \( X, \delta \).

The general case is then a simple extension.

5.3.3. Definition. For every operator \( F \) from \( \text{dom} F = \prod [p_1 \rightarrow q_1] \times \cdots \times [p_m \rightarrow q_m] \) to \( [r_1 \rightarrow s_1] \times \cdots \times [r_n \rightarrow s_n] \) a compensation structure is an \( n \)-tuple \( \{(\Pi_j, w_j, I_j) : j = 1, \ldots, n\} \) such that for each \( j = 1, \ldots, n \), the \( j \)th triple is a compensation structure for \( P_j F \).

5.4. Compensating for subequivalence: proofs

In this section we show that all operators in \( \text{MO} \) have a compensation structure.

5.4.1. Lemma. The constant, projection, disjoint union, identity, composition and link operators have a compensation structure.

Proof. We first exhibit, for each of these operators, the required functions.

1. Modelling and causal constant operators: \( \Pi = \emptyset \) set. Choose \( w \) and \( I \) also to be empty.

2. Projection operators: \( \lambda(f_1, \ldots, f_m).f_i \) defines a projection operator from \( [p_1 \rightarrow q_1] \times \cdots \times [p_m \rightarrow q_m] \) to \( [p_i \rightarrow q_i] \). Here

\[
\Pi = \{e\},
\]

\[
w(e) = i,
\]

\[
I(f_1, \ldots, f_m, e, X, \delta) = X, \quad \text{for all } X \in p_i.
\]
(3) **Disjoint union:** Let \( \text{DU} \) be defined as in Section 4.3. Again \( n = 1 \), and

\[
\Pi = \{ L, R \},
\]

\[
w(L) = 1, \quad w(R) = 2.
\]

Let \( X = (h_1, \ldots, h_p, h_{p+1}, \ldots, h_{p+r}) \). Then for all \( f_1, f_2 \),

\[
I((f_1, f_2), L, X, \delta) = (h_1, \ldots, h_p), \quad I((f_1, f_2), R, X, \delta) = (h_{p+1}, \ldots, h_{p+r}).
\]

(4) **Identity:**

\[
\Pi = \{ e \},
\]

\[
w(e) = 1,
\]

\[
I(f, e, X, \delta) = X.
\]

(5) **Composition:**

\[
\Pi = \{ L, R \},
\]

\[
w(L) = 2, \quad w(R) = 1,
\]

\[
I((f_1, f_2), L, X, \delta) = X, \quad I((f_1, f_2), R, X, \delta) = f_2(X, \lambda(\delta)).
\]

(6) **Link operators:** Let \( \text{LINK}_i \) be defined as in Section 4.3. Again \( n = 1 \), and

\[
\Pi = \{ e \},
\]

\[
w(e) = 1,
\]

\[
I(f, e, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p, \delta) = (x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta),
\]

where \( H \), as defined as in Definition 4.3.7, is the least fixed point of

\[
H = [f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)]_k.
\]

It is straightforward to check that the above functions have the required properties. The only nontrivial case is to verify Definition 5.3.2(f) for the link operators. We do this now. On the basis of the assumption that \( \delta' = o(e) \) compensates \( f \) to \( g \) on \( (x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p), \delta \), we have to show that \( \delta' \) compensates \( \text{LINK}_i f \) to \( \text{LINK}_k g \) on \( (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p), \delta \). That is, we assume that

\[
[f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)]_r \geq [g(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta')]_r
\]

for \( r = 1, \ldots, k-1, k, k+1, \ldots, q \),

and show that

\[
[f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)]_r \geq [g(x_1, \ldots, x_{i-1}, H', x_{i+1}, \ldots, x_p, \delta')]_r
\]

for \( r = 1, \ldots, k-1, k+1, \ldots, q \).
where \( H' \) is the least fixed point of
\[
H' = [g(x_1, \ldots, x_{i-1}, H', x_{i+1}, \ldots, x_p, \delta')]_k.
\]

It suffices to show that \( H \geq H' \). Now
\[
H_0 = [g(x_1, \ldots, x_{i-1}, \bot, x_{i+1}, \ldots, x_p, \delta')]_k
\leq [g(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta')]_k
\leq [f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)]_k
= H.
\]

Inductively, we assume that \( H_i \leq H \) and show that \( H_{i+1} \leq H \).
\[
H_{i+1} = [g(x_1, \ldots, x_{i-1}, H_i, x_{i+1}, \ldots, x_p, \delta')]_k
\leq [g(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta')]_k \quad \text{by inductive assumption}
\leq [f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)]_k
= H.
\]

It follows that \( H' \leq H \).

5.4.2. Lemma. If \( O_1, \ldots, O_n \) have compensation structures then so does \([O_1, \ldots, O_n]\).

Proof. Immediate from definitions. □

5.4.3. Lemma. If \( F, G \) have compensation structures then so does \( FG \).

Proof. See Appendix.

Theorem 5.4.4 follows from the above lemmas.

5.4.4. Theorem. Every operator in \( MO \) has a compensation structure.

We are also interested in the following “infinite history” property of certain operators and their compensation structures. We shall show that all operators in \( MO \), with compensation structures defined as above, have the infinite history property.

5.4.5. Definition. Let \( F \) be an operator from \([p_1 \rightarrow q_1] \times \cdots \times [p_m \rightarrow q_m]\) to \([r_1 \rightarrow s_1] \times \cdots \times [r_n \rightarrow s_n]\) with compensation structure \( \{(\Pi_j, w_j, I_j) : j = 1, \ldots, n\} \). We say \( F \) has the infinite history property if, for all \( 1 \leq j \leq n \), for all \( (f, \pi, X, \delta) \in \text{dom } I_j \); if \( f \) is causal and \( X \) is infinite (at all components), then \( I_j(f, \pi, X, \delta) \) is infinite (at all components).
5.4.6. **Lemma.** The constant, projection, disjoint union, identity and composition operators, with compensation structures as defined in Lemma 5.4.1, have the infinite history property.

**Proof.** Straightforward, noting that causal functions map infinite histories to infinite histories. □

5.4.7. **Lemma.** If $F$ and $G$ are dominance preserving and have the infinite history property, then $FG$, with compensation structure as defined in Lemma 5.4.3, has the infinite history property.

**Proof.** Straightforward, noting that dominance preserving operators map causal functions to causal functions. □

5.4.8. **Lemma.** If $G$ is modelling and has the infinite history property, then $\text{LINK}_i G$, with compensation structure as defined in Lemma 5.4.3, has the infinite history property.

**Proof.** See Appendix.

Theorem 5.4.9 follows from the above lemmas and a simple induction.

5.4.9. **Theorem.** All operators in $\mathbf{MO}$, with compensation structures defined in Lemmas 5.4.1–5.4.3, have the infinite history property.

5.5. **Preservation of behaviour equivalence: proofs**

Here we give a proof of Lemma 5.2.3, hence completing the proof that modelling contexts preserve behaviour equivalence.

**Proof of Lemma 5.2.3.** Extending the notation used in this lemma, we write $y=(y_1, \ldots, y_n)$, $z=(z_1, \ldots, z_n)$ and $F_j-P_j F, j=1, \ldots, n$. We also write $f=(f_1, \ldots, f_m)$ and $g=(g_1, \ldots, g_m)$.

To show $y \ll z$, we show that $y_j \ll z_j, j=1, \ldots, n$. That is we show for all infinite $X$ and all $\delta_j$ that there is $\delta_j'$ such that $y_j(X, \delta_j) = z_j(X, \delta_j)$.

Now, $z$ is the least fixed point of $z = F(g, z)$; so, each $z_j$ is a modelling process [Theorem (4.2.1)]. Also, as $X$ is infinite in all components, each $z_j(X, \delta_j)$ is infinite, so it is enough to show that $y_j(X, \delta_j) \geq z_j(X, \delta_j)$. We write $z = \text{lub } z_k$, where $z_0 = \bot$ and for all $k \geq 0, z_{k+1} = F(g, z_k)$. We write $z_k = (z_{1,k}, \ldots, z_{n,k})$. It is enough to show that for all $j=1, \ldots, n$, all infinite $X$ and all $\delta_j$ there is $\delta_j'$ such that for all $k \geq 0, y_j(X, \delta_j) \geq z_{j,k}(X, \delta_j)$. Refer to the example diagrams shown in Fig. 6.
To construct $\delta'_j$, we construct for each $X$ and $\delta_j$ an increasing sequence $(\gamma_{j,k}(X, \delta_j))$ of elements of $\Omega_{AS}$ such that for all total $\delta''_j \geq p \gamma_{j,k}(X, \delta_j), \gamma_j(X, \delta_j) \geq z_{k,h}(X, \delta''_j)$.

For then $(\gamma_{j,k}(X, \delta_j))$ has an upper bound $\gamma_j$ say [Lemma (2.3.10)] and we can choose any total strongly fair $\delta''_j \geq p \gamma_j$. We shall often abbreviate $(\gamma_{j,k}(X, \delta_j))$ to $\gamma_{j,k}$ where $X$ and $\delta_j$ are clear from the context. To construct $(\gamma_{j,k})$ we consider a compensation structure for $F_j$, say $(\Pi_j, w_j, I_j), j = 1, \ldots, n$. First we introduce some notation.

Intuitively, we write $u_j$ for the set of positions at which any of the $f$ (or $g$) parameters may be substituted in $F_j$. Likewise $v_j$ is the set of positions at which any of the $y$ (or $z_k$) parameters may be substituted. More precisely, we write $u_j = \bigcup_{i=1}^{m} w_j^{-1}(i), v_j = \bigcup_{i=m+1}^{m+n} w_j^{-1}(i)$.

We also consider $f$ and $g$. Now $f \ll g$, i.e. $f_i \ll g_i, i = 1, \ldots, m$. Also, as $f$ is modelling and $F$ is in MO, then by the infinite history property of $F$, $I_j((f, y), \pi, X, \delta_j)$ is infinite. Then, by the definition of compensation there is $\sigma_j: u_j \rightarrow \Delta$ such that for all $X$ and $\delta_j$, all $i = 1, \ldots, m$ and all $\pi \in w_j^{-1}(i), \sigma_j(\pi)$ compensates $f_i$ to $g_i$ on $I_j((f, y), \pi, X, \delta_j), \text{sub}(\pi, \delta_j)$.

Intuitively, for each position $\pi$ of an $f$ in $F_j, \sigma_j(\pi)$ gives the oracle value required to compensate the $f$ at that position to a $g$ at that position. Now, to define $(\gamma_{j,k})$ by recursion on $k$ we also define, for $k \geq 0$, parallel updates $\sigma_{j,k}(X, \delta_j)$ with domain $v_j$.

Intuitively, for each $f$ position, $\sigma_j$ is used to update $\delta_j$, and for each $y$ position, $\sigma_{j,k}$ is used to update $[\sigma_j] \delta_j$. The definitions of $\sigma_{j,k}$ and $\gamma_{j,k}$ are mutually recursive.
Intuitively, for each position of a $y$ parameter in $F_j$, $\sigma_{j,k}(X, \delta_j)$ specifies the class of oracles which may be used to compensate the $y$ at that position to a $z_{i,k}$ at that position. More precisely, $\sigma_{j,k}(X, \delta_j)$ is defined, for all $i = 1, \ldots, n$ and all $\pi \in \omega_j^{-1}(i + m)$, by

$$(\sigma_{j,k}(X, \delta_j))(\pi) = \gamma_{i,k}(I_j(f, y), \pi, X, \delta_j), \text{sub}(\pi, \delta_j)).$$

The definition of $(\gamma_{j,k})$ is:

$$\gamma_{j,0}(X, \delta_j) = \bot, \quad \gamma_{j,k+1}(X, \delta_j) = [\sigma_{j,k}(X, \delta_j)] [\sigma_j] \delta_j, \quad k \geq 0,$$

for all $X$ and $\delta_j$. Now we prove each of the following by induction on $k$, to complete the argument. For all $j = 1, \ldots, n$, for all $X$ and $\delta_j$ and all $k \geq 0$.

(i) if $k > 0$, $\gamma_{j,k}(X, \delta_j) \supseteq \gamma_{j,k-1}(X, \delta_j)$.

(ii) for all total $\delta'_j \supseteq \gamma_{j,k}(X, \delta_j)$, $\gamma_j(X, \delta_j) \supseteq z_{j,k}(X, \delta'_j)$.

Consider (i). For $k = 1$, it is true from Lemma 2.3.13. For $k > 1$, from the definition of $\gamma_{j,k}$ and Lemma 2.3.12 it is enough to show that for all $\pi \in \omega_j$, $\gamma_{j,k-1}(\pi) \supseteq \gamma_{j,k-2}(\pi)$.

This follows from the definition of the $\sigma_{j,k}$'s and the inductive hypothesis.

Finally, consider (ii). The case $k = 0$ is trivially true since $z_{j,k} = 1$. For $k \geq 1$ consider arbitrary total $\delta'_j \supseteq \gamma_{j,k}(X, \delta)$. From the definitions of $z_{j,k}$ and $y_j$ it is enough to show that

$$(F_j(f, y))(X, \delta) \supseteq (F_j(g, z_{k-1}))(X, \delta'_j), \quad \text{for all total } \delta'_j \supseteq \gamma_{j,k}(X, \delta).$$

For that it is enough to show that $\delta'_j$ compensates $F_j(f, y)$ to $F_j(g, z_{k-1})$ on $X, \delta_j$.

To achieve that, using Definition 5.3.2(f) we find an update $\sigma'_j$ with domain $P_j$ such that $[\sigma'_j] \delta_j = \delta'_j$ and verify that

(a) for all $i = 1, \ldots, m$, and all $\pi \in \omega_j^{-1}(i), \sigma'_j(\pi)$ compensates $f_i$ to $g_i$ on $I_j(f, y), \pi, X, \delta_j), \text{sub}(\pi, \delta_j)$.

(b) for all $i = 1, \ldots, n$, and all $\pi \in \omega_j^{-1}(i + m), \sigma'_j(\pi)$ compensates $y_i$ to $z_{i,k-1}$ on $I_j(f, y), \pi, X, \delta_j), \text{sub}(\pi, \delta_j)$.

We choose for all $\pi \in P_j, \sigma'_j(\pi) = \text{sub}(\pi, \delta'_j)$. From the definition of $\delta'_j$ it is clear that $[\sigma'_j] \delta_j = \delta'_j$. Moreover $\delta'_j \supseteq \gamma_{j,k}$ so the definition of $\gamma_{j,k}$ makes (a) clear. To show (b) we have to show that for $i = 1, \ldots, n$

$$y_i(I_j(f, y), \pi, X, \delta_j), \text{sub}(\pi, \delta_j)) \supseteq z_{i,k-1}(I_j((f, y), \pi, X, \delta_j), \text{sub}(\pi, \delta'_j)).$$

This will follow from inductive hypothesis (ii) provided we can show that

$$\text{sub}(\pi, \delta'_j) \supseteq \gamma_{i,k-1}(I_j(f, y), \pi, X, \delta_j), \text{sub}(\pi, \delta_j)).$$

By definition of $\delta''$, $\text{sub}(\pi, \delta'_j) \supseteq \gamma_{j,k}(X, \delta_j))$

$$= (\sigma_{j,k-1}(X, \delta_j))(\pi)$$

$$= \gamma_{i,k-1}(I_j((f, y), \pi, X, \delta_j), \text{sub}(\pi, \delta_j)), as required to complete the proof.
6. The nondeterministic level of the model

6.1. Basic definitions

We can now define the basic structure of our model of nondeterministic data flow. Recall that waits do not occur in nondeterministic histories.

6.1.1. Definition. A nondeterministic process with $m$ inputs and $n$ outputs is defined to be a behaviour equivalence class of deterministic modelling processes of type $[m \rightarrow n]$.

The output behaviours of a nondeterministic process $P$ with $m$ nondeterministic inputs $X_1, \ldots, X_n$ are obtained as follows.

6.1.2. Definition. First the function $E$, which strips out waits from histories, is defined as follows.

$$E(\perp) = \perp.$$  
$$E(a.X) = a.E(X), \quad \text{if } a \neq \tau.$$  
$$E(a.X) = E(X), \quad \text{if } a = \tau.$$  

Note: $E$ is not a modelling function. It is used to specify the computation of nondeterministic behaviours.

6.1.3. Definition. For a nondeterministic history vector of histories $X_1, \ldots, X_n$ we define the set $mh(X_1, \ldots, X_n)$ of deterministic model histories by

$$mh(X_1, \ldots, X_n) = \{(X_1', \ldots, X_n') | X_i = E(X'_i), X_i \text{ infinite, } i = 1, \ldots, n\}.$$  

6.1.4. Definition. The set of nondeterministic output behaviours for $P$, given nondeterministic input histories $X_1, \ldots, X_n$, is now defined as follows. Choose a representative process in the equivalence class of processes modelling $P$. Call this process $f$. The set of nondeterministic output behaviours is:

$$\{E(f(X'_1, \ldots, X'_n, \delta)) | (X'_1, \ldots, X'_n) \in mh(X_1, \ldots, X_n), \delta \in A\}.$$  

6.1.5. Lemma. The set of nondeterministic output behaviours for $P$, given $X_1, \ldots, X_n$, does not depend on which behaviour equivalence class representative is chosen.

Proof. This follows from the definition of equivalence since all model histories are infinite. $\square$

6.2. Nondeterministic network construction

For each of the modelling deterministic network construction operators there is a corresponding nondeterministic network construction operator. We write
for typical nondeterministic processes and \( m(P_1), m(P_2) \) for representative members of the equivalence classes \([P_1], [P_2]\). Further, we write \( P \) for the function which maps a deterministic modelling process to its equivalence class and \( \text{DO} \) for a deterministic modelling network construction operator. For each \( \text{DO} \) there is a corresponding nondeterministic operator \( \text{NDO} \) defined by

\[
\text{NDO}(P_1, \ldots, P_n) = P(\text{DO}(m(P_1), \ldots, m(P_n))).
\]

**6.2.1. Proposition.** \( \text{NDO} \) is well defined, i.e. if \( D_i \equiv E_i, i = 1, \ldots, n \), are modelling processes, then \( P(\text{DO}(D_1, \ldots, D_n)) = P(\text{DO}(E_1, \ldots, E_n)) \).

**Proof.** Corollary of 5.2.2. \( \square \)

*Note:* This result together with 4.5.1 and 5.1.5 shows that construction of nondeterministic networks is also associative. It also follows that anomalies of the Brock–Ackerman type cannot arise in our model.

### 6.3. Fairness

#### 6.3.1. Fair Merge

We show that the model of fair merge given in the introductory example does indeed model the nondeterministic behaviours of a nondeterministic fair merge. We write \( \text{FM} \) for our model of fair merge:

\[
\text{FM}(X_0, X_1, \delta) = \text{dmerge}(\text{inject}(X_0, \text{Dist}_1(\delta)), \text{inject}(X_1, \text{Dist}_2(\delta))).
\]

To show our \( \text{FM} \) models fair-merge we have to show that for all nondeterministic inputs \( X_0, X_i \) and all deterministic models \( X_i \) of \( X_i(i = 0, 1) \) and all fair merges \( m' \) of \( X_0, X_i \), there is a strongly fair \( \delta \) such that

\[
m' = E(\text{FM}(X_0, X_1, \delta)).
\]

For convenience we show more: that for all fair merges \( m \) of \( X_0, X_i \) there is some strongly fair \( \delta \) such that

\[
E(m) = E(\text{FM}(X_0, X_1, \delta)). \tag{1}
\]

This is stronger since, evidently, for every fair merge \( m' \) of \( X_0, X_i \) there is a fair merge \( m \) of \( X_0, X_i \) such that \( E(m) = m' \). To show (1) it is enough, from the definition of \( \text{FM} \), to construct \( \delta_0, \delta_1 \) strongly fair such that

\[
m' = E(\text{dmerge}(\text{inject}(X_0, \delta_0), \text{inject}(X_1, \delta_1)))
\]

since we may take the required \( \delta \) as \( U(\delta_0, \delta_1) \).

The algorithm below constructs \( \delta_0, \delta_1 \) as required. We describe it using the following notation.
(1) We write $R$ for an infinite sequence of 0's and 1's which specifies a decomposition of $m$ into $X_0$ and $X_1$. That is, $X_i$ is the subsequence $(m_k)$ of $m$ such that $R_k = i, i = 0, 1$. Since each $X_i$ is infinite, $R$ is fair.

(2) We write $E_i, i = 0, 1$, for some enumeration of the alternating subsequences of $\delta_i$ such that each alternating subsequence of $\delta_i$ occurs in $E_i$ infinitely often. Algorithmic representation of $E_i$ is not difficult and is omitted. We may write $E_{i,k}$ for the $k$th alternating sequence of $E_i$.

In each step $j$ of the algorithm we guarantee that the next nonwait contribution to output from an input stream is taken from $X_{R_j}$, the input stream specified by the $j$th element of $R$. At the same time we ensure that at least one 1 and one 0 are placed in the subsequence of $\delta_{R_j}$ specified by the next element of $E_{R_j}$.

Since $R$ is fair, all terms of both $E_0$ and $E_1$ will be considered. Since each subsequence of $\delta_i$ occurs infinitely often in $E_i$, we ensure that all alternating subsequences of $\delta_i$ are fair, and hence, that $\delta_i$ is strongly fair.

Recall that 1's inserted into $\delta_i$ cause the injection of waits into the $i$th input stream $X_i$ of the deterministic alternating merge.

Write $d_0, d_1$ for the segments of $\delta_0, \delta_1$ so far constructed and $n(j, i)$ for the number of i's in $R_1, ..., R_{j-1}$ ($i = 0, 1$).

The $j$th stage of the algorithm extends both $d_0$ and $d_1$ by appending 1's until a 1 has been placed in the subsequence $E_{R_j, n(j, R_j)}$ and until the next extension of $d_0$ and $d_1$ would again extend the subsequence of $E_{R_j, n(j, R_j)}$. Then $d_{R_j}$ is extended by a 0 and the other $d$ is extended by a 1.

### 6.3.2. A fair sequence generator

From Section 6.3.1 it is clear that $\text{FS}$ given by

$$\text{FS}(\delta) = \text{FM}(0^\omega, 1^\omega, \delta)$$

will generate all fair sequences of 0's and 1's, i.e.

$$\{E(\text{FS}(\delta)) | \delta \in A\} = \{s | s \text{ is a fair sequence of 0's and 1's}\}.$$  

### 6.3.3. An unfair sequence generator

First define a deterministic modelling process $\text{AD}$ which discards every second nonwait input.

$$\text{AD}(X, \delta) = \tau.\text{AD}'(X, \delta, 0).$$  

$$\text{AD}'(\bot, \delta, p) = \bot.$$  

$$\text{AD}'(\tau.X, \delta, p) = \tau.\text{AD}'(X, \delta, p).$$  

$$\text{AD}'(a.X, \delta, 0) = a.\text{AD}'(X, \delta, 1), \ a \neq \tau.$$  

$$\text{AD}'(a.X, \delta, 1) = \tau.\text{AD}'(X, \delta, 0), \ a \neq \tau.$$
Now, using FS from Section 6.3.2, we define

\[ US(\delta) = AD(FS(Dist_1(\delta)), Dist_2(\delta)). \]

It is easy to see that

\[ \{ E(US(\delta)) \mid \delta \in A \} = \{ s \mid s \text{ is an infinite sequence of 0's and 1's} \}. \]

### 6.4. Extensionality of the model

To motivate our approach to extensionality we observe that although waits do not appear explicitly in nondeterministic histories, they can strongly influence the output histories of a nondeterministic process. As mentioned above, it is possible to consider a restriction of the present theory in which oracles are used only to control the injection of waits into data streams. In that subtheory, all the nondeterministic behaviours can be regarded as manifestations of underlying waits.

There is nothing in our definitions to exclude nondeterministic processes which provide evidence at the nondeterministic level of wait items present in deterministic histories. Further, while modelling processes are required to propagate waits (in the sense of Definition 3.2.4), they can respond to waits in other ways as well.

We now exhibit a modelling process which is a general “wait visualiser”. It has one nonoracle input and two outputs. This process ignores its oracle input. We define the two outputs using functions \( \text{viz}_1, \text{viz}_2 \).

\[ \text{viz}_1(X, \delta) = \tau \cdot \text{viz}_1'(X, \delta). \]
\[ \tau \cdot \text{viz}_1'(X, \delta) = \begin{cases} 0 \cdot \text{viz}_1'(X, \delta), & \text{if } \tau = \tau, \\ 1 \cdot \text{viz}_1'(X, \delta), & \text{otherwise}. \end{cases} \]
\[ \text{viz}_2(X, \delta) = \tau \cdot X. \]

The second output function \( \text{viz}_2 \) is just a modelling identity function. The first output function \( \text{viz}_1 \) encodes the pattern of runs of waits in the input stream by putting out a 0 if it encounters a wait and 1 otherwise. Note that

\[ (E(\text{viz}_1(X, \delta)) = E(\text{viz}_1(X', \delta'))) \text{ and } E(\text{viz}_2(X, \delta)) = E(\text{viz}_2(X', \delta')) \]

iff \( X = X' \).

At the nondeterministic input–output behaviour level the above process will appear highly erratic. Nevertheless, such processes exist in our theory. With respect to the class of nondeterministic processes modelled in our framework, our model is extensional in the sense of the following theorem. Recall that nondeterministic processes are observational equivalence classes of deterministic processes.

### 6.4.1. Extensionality theorem

If \( P_1, P_2 \) are nondeterministic processes and \( P_1 \neq P_2 \), then there exists a nondeterministic context \( \text{NDO} \) such that \( \text{NDO}(P_1) \) and \( \text{NDO}(P_2) \) exhibit different nondeterministic output behaviours in response to the same nondeterministic input.
6.4.2. Corollary. If nondeterministic processes $P_1, P_2$ exhibit the same nondeterministic input–output behaviours in all nondeterministic contexts, then $P_1 = P_2$.

Proof. We assume for simplicity that $P_1, P_2$ map one input to one output. The argument generalises simply to vectors of inputs and outputs. We write $f, g$ for representatives of the equivalence classes $[P_1], [P_2]$. Since $P_1 \neq P_2$, $f$ is not behaviour-equivalent to $g$. Assume without loss of generality that $f$ is not subequivalent to $g$. Therefore there exist infinite $Y'$ and $\delta \in \Delta$ such that there does not exist any $\delta' \in \Delta$ with $f(Y', \delta) = g(Y', \delta')$. We write $Y = E(Y')$.

Figure 7 sketches a nondeterministic network scheme in which substitutions of $f$ and $g$ respectively for $P$ will yield networks with different nondeterministic input–output behaviours. The causal process fanout is defined by

$$\text{fanout}(X) = (X, X).$$

Let us write $NF$ for the result of substituting $f$ and $NG$ for the result of substituting $g$. We have labelled the output ports 1 through 4. The network $NF$ can produce a vector of nondeterministic output behaviours from nondeterministic input $Y$ which $NG$ cannot. Specifically, one of the output vectors of $NF$ will arise from an input to $f$ (in the deterministic model) of $Y'$ and $\delta$. The network $NG$ cannot produce this behaviour since if the nondeterministic outputs on ports 1 and 2 match, the nondeterministic outputs on ports 3 and 4 cannot.

7. A metric space framework for the analysis of modelling networks

Park has suggested the use of a metric space framework for the treatment of fixed points in our system. We outline below a metric space approach which has some
technical advantages compared to the partial order approach. Proofs are omitted here; they may be found in [7]. We have used partial orders in the earlier work because their relationship to operational interpretations is more widely known, e.g. through Kahn and Faustini’s work in deterministic data flow [4, 5].

7.1. Metrics of histories and oracles

Write $D^n$ for the set of denumerably infinite sequences of elements of $D$ which include infinitely many waits. (The restriction that infinitely many waits appear can be removed if desired). In this section we call elements of $D^n$ metric histories. For $n \geq 1$, elements of $(D^n)^n$ may be called metric history vectors. We define a metric on $D^n$ as follows. We write $\text{glb}(h_1, h_2)$ for the (in general finite) greatest lower bound of $h_1, h_2$ in the prefix ordering. We use the notation $\#s$ to denote the length of a sequence $s$.

We first define the agreement of two histories $h_1$ and $h_2$, which we write as $a(h_1, h_2)$.

**7.1.1. Definition.** For all pairs of histories $h_1, h_2$, we define their agreement $a(h_1, h_2)$ to be the number of waits in $\text{glb}(h_1, h_2)$. Formally,

$$a(h_1, h_2) = \# \{ i : 1 \leq i \leq \# \text{glb}(h_1, h_2) : \text{glb}(h_1, h_2)_i = \tau \}.$$

In other words, referring to the operational interpretation of waits discussed in Section 3, the agreement of $h_1, h_2$ is the period of time before divergence.

**7.1.2. Definition.** We now define the distance between histories $h_1$ and $h_2$, written as $d(h_1, h_2)$:

$$d(h_1, h_2) = 2^{-a(h_1, h_2)}.$$

It may be verified that the distance so defined is a metric on $D^n$. Indeed, it is an ultrametric, i.e. it satisfies the strong triangle inequality for histories $h_1, h_2, h_3$:

$$d(h_1, h_3) \leq \max(d(h_1, h_2), d(h_2, h_3)).$$

Further, $(D^n, d)$ is a complete metric space. We extend the metric to metric history vectors as follows.

**7.1.3. Definition.** Let $h = (h_1, \ldots, h_n)$ and $k = (k_1, \ldots, k_n)$ be $n$-tuples of metric histories. We define

$$d(h, k) = \max(d(h_1, k_1), \ldots, d(h_n, k_n)).$$

The distance so defined is also an ultrametric and $((D^n)^n, d)$ is a complete metric space. In accordance with our approach of treating oracles purely as sources of nondeterminism, we have not included waits in oracle sequences. We now define an appropriate metric on strongly fair oracles.
7.1.4. Definition. Let $\delta, \gamma \in \Delta$. We define the distance $d(\delta, \gamma)$ between $\delta, \gamma$:

$$d(\delta, \gamma) = 2^{-\# \text{glb}(\delta, \gamma)}.$$ 

This is also an ultrametric on $\Delta$. However, $(\Delta, d)$ is not complete. It may be verified that $\text{Dist}$ is continuous on $\Delta \to \Delta \times \Delta$.

7.2. Modelling functions as contractions

Corresponding to our modelling functions are the contraction maps with respect to histories. In fact we may take a uniform contraction constant of $\frac{1}{2}$.

7.2.1. Definition. The metric modelling functions on $(D^\omega)^m \times \Delta \to (D^\omega)^n$ are the functions $f$ such that

1. for all histories $x$ and $y$, $d(f(x, \delta), f(y, \delta)) \leq \frac{1}{2}d(x, y)$.
2. $f$ is continuous with respect to its oracle input.

Contraction maps have unique fixed points. For each metric modelling function $f$ and for each oracle $\delta$, writing $g(x) = f(x, \delta)$, the unique solution to $x = g(x)$ is $\lim_{n \to \infty} g^n(y)$, for arbitrary $y$ in the domain of $g$.

Remark: An alternative approach is possible using the metric on histories defined by

$$d'(x, y) = 2^{-\# \text{glb}(x, y)}.$$ 

In this case the corresponding modelling functions would be length-increasing with respect to nonoracle input. This approach is very close to that of Park [10].

7.2.2. Definition. The metric causal functions on $(D^\omega)^m \times \Delta \to (D^\omega)^n$ are the functions $f$ such that

1. for all histories $x$ and $y$, $d(f(x, \delta), f(y, \delta)) \leq d(x, y)$.
2. $f$ is continuous with respect to its oracle input.

Denote the space of continuous functions from $(D^\omega)^m \times \Delta$ to $(D^\omega)^n$ by $[m \to n]$. Let $M_{m,n}$ denote the subspace of metric modelling functions in $[m \to n]$. We next define a metric on $[m \to n]$.

7.2.3. Definition. Let $f_1, f_2 \in [m \to n]$. The distance $d(f_1, f_2)$ between $f_1$ and $f_2$ is defined by

$$d(f_1, f_2) = \sup_{x, \delta} d(f_1(x, \delta), f_2(x, \delta)).$$
7.2.4. **Lemma.** The space $M_{m,n}$ with the metric given forms a complete metric space.

We extend the definitions to product function spaces.

7.2.5. **Definition.** Let $f=(f_1, \ldots, f_n)$ and $g=(g_1, \ldots, g_n)$ belong to $[p_1 \to q_1] \times \cdots \times [p_n \to q_n]$. Then we define

$$d(f,g) = \max(d(f_1,g_1), \ldots, d(f_n,g_n)).$$

7.3. **Modelling operators**

An operator is a map from $[p_1 \to q_1] \times \cdots \times [p_n \to q_n]$ to $[r_1 \to s_1] \times \cdots \times [r_n \to s_n]$. We now define the metric modelling operators.

7.3.1. **Definition.** An operator $F$ is a *metric modelling* operator if

1. it maps metric modelling functions to metric modelling functions, and
2. it is contracting, i.e. for all modelling $f$ and $g$, $d(F(f), F(g)) \leq d(f, g)$.

7.3.2. **Definition.** An operator is called a *nonexpansive modelling* operator if

1. it maps metric modelling functions to metric modelling functions, and
2. it is nonexpansive, i.e. for all modelling $f$ and $g$, $d(F(f), F(g)) \leq d(f, g)$.

Metric modelling operators have unique modelling fixed points. The unique solution to $f = F(f)$ may be calculated by taking the limit of $F^n(y)$, where $y$ is an arbitrary modelling function of the appropriate type. We now define a specific class of metric modelling operators using the vocabulary of basic operators introduced in Section 4.3. The definitions of the operators given in Section 4.3 may be carried over unchanged to the metric space framework, except for that of the link operator. We give that definition now.

7.3.3. **Definition (link operator).** Let $f \in M_{p,q}$ and let $i \in \{1, \ldots, p\}, k \in \{1, \ldots, q\}$. For $r=1, \ldots, k-1, k+1, \ldots, q$

$$\text{LINK}_i(f)(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p, \delta)_r = [f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)]_r,$$

where $H$ is the unique fixed point of

$$H = [f(x_1, \ldots, x_{i-1}, H, x_{i+1}, \ldots, x_p, \delta)]_k,$$

noting that for all $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p, \delta), g$ defined by

$$g(h) = [f(x_1, \ldots, x_{i-1}, h, x_{i+1}, \ldots, x_p, \delta)]_k$$

is a contraction.
7.3.4. Definition. First we define a class $\text{NE}$ of nonexpansive modelling operators. The set $\text{NE}$ is the least set of operators closed under the following conditions:

1. The constant modelling operators and the projection operators belong to $\text{NE}$.
2. $\text{DU},\text{Id},\text{Comp},\text{LINK}\in\text{NE}$.
3. For all $G_1, \ldots, G_m\in\text{NE}$, $[G_1, \ldots, G_m]\in\text{NE}$.
4. For all $G_1, G_2\in\text{NE}$, $G_1G_2\in\text{NE}$.

We now define a class $\text{MO}$ of metric modelling operators.

7.3.5. Definition. The set $\text{MO}$ is the least set of operators closed under the following conditions:

1. The constant modelling operators belong to $\text{MO}$.
2. For all $G\in\text{MO}$ and $D\in\text{NE}$, $\text{Comp}[G,D]\in\text{MO}$.
3. For all $G\in\text{MO}$ and for every causal constant operator $C$, $\text{Comp}[G,C]\in\text{MO}$ and $\text{Comp}[C,G]\in\text{MO}$.
4. For all $G_1, \ldots, G_m\in\text{MO}$, $[G_1, \ldots, G_m]\in\text{MO}$.
5. For all $G_1, G_2\in\text{MO}$ and $D\in\text{NE}$, $G_1D, DG_1, G_1G_2\in\text{MO}$.

7.4. Behaviour equivalence

Definitions of behaviour equivalence and subequivalence are as in Section 5.1. In the metric framework, we use the following notion of compensation.

7.5.1. Definition. We say that $\delta'$ compensates $f$ to $g$ to within $\varepsilon$ on $X,d$ if $d(f(X,d),g(Y,d'))<\varepsilon$.

In the metric space framework we define compensation structures as follows.

7.5.2. Definition. For every operator $F$ from $\text{dom } F = \prod_{i=1}^{m-1} p_i \to \prod_{i=m}^{2m} q_i$ to $\prod_{i=1}^{m} r$, a compensation structure for $F$ is a tuple $(\Pi, w, I, c)$ with the following properties:

(a) $\Pi$ is a finite independent set of positions.
(b) $w$ is a function from $\Pi$ to $\{1, \ldots, m\}$.
(c) $I$ is a function from $\text{dom } I = \text{dom } F \times \Pi \times r \times \Delta$ to $\bigcup_{i=1}^{m} p_i$.
(d) $c$ is a real number $\geq 0$, called the compensation constant.
(e) For all $(f, \pi, X, \delta)\in\text{dom } I, I(f, \pi, X, \delta)\in p_{w(\pi)}$.
(f) For all updates $\sigma : \Pi \to \Delta$, if for all $\pi \in \Pi$, $\sigma(\pi)$ compensates $f_{w(\pi)}$ to $g_{w(\pi)}$ to within $\varepsilon$ on $I(f, \pi, X, \delta), \text{sub}(\pi, \delta)$, then $[\sigma] \delta$ compensates $F(f)$ to $F(g)$ to within $\varepsilon c$ on $X, \delta$.

The general case is then a simple extension.
7.5.3. Definition. For every operator \( F \) from \([p_1 \rightarrow q_1] \times \cdots \times [p_m \rightarrow q_m]\) to \([r_1 \rightarrow s_1] \times \cdots \times [r_n \rightarrow s_n]\), a compensation structure is an \( n \)-tuple

\[
\{(\Pi_j, w_j, l_j, c_j) : j = 1, \ldots, n\}
\]

such that for each \( j = 1, \ldots, n \), the \( j \)th tuple is a compensation structure for \( P_j F \). We define the compensation constant of a tuple of compensation structures to be the maximum of the compensation constants of its component compensation structures.

We can show that the metric modelling operators of \( \textbf{MO} \) as defined in Section 7.3 have compensation structures with compensation constant less than or equal to \( \frac{1}{2} \). A lemma corresponding to Lemma 5.2.3 may be proved using similar proof ideas.

8. Discussion and conclusions

We have presented a model of nondeterministic data flow processes based on the Kahn fixed-point semantics for deterministic data flow. Nondeterministic output behaviours may be calculated by calculating deterministic outputs on a class of deterministic inputs. We have shown that the semantics is abstract (Theorem 6.2.1) and extensional (Theorem 6.4.1).

As stated earlier, the present work is founded on [10]. In this paper Park developed a model of nondeterministic data flow based on modelling nondeterministic processes by deterministic processes operating on infinite streams containing "hiatons" (our "waits") and processes having oracle inputs. It is clear that the present work is indebted to Park. We summarise now ways in which the present paper refines and develops the work of [10].

(1) Park restricts his detailed analysis to networks composed of "deterministic" processes (i.e. processes whose nondeterministic behaviours are insensitive to the presence of waits) and nondeterministic merge processes. Our analysis is more general, covering a wide class of deterministic modelling processes and thus, a wide class of nondeterministic processes.

(2) Park uses \( >\)-functions for the deterministic and merge processes as his class of modelling processes. These functions always produce output with length greater than their input. Our class of modelling functions is less restrictive.

(3) In [10] network construction results in networks with a variable number of oracle inputs. Our mechanism for distributing oracle inputs from a single source gives a modular network construction principle, without loss of generality.

(4) Because of point (3) it is not clear how Park's approach can deal with recursively defined networks. Our work includes recursively defined networks (Theorems 4.2.1 and 5.2.1).

(5) The model of [10] is not extensional.
Appendix

This appendix includes several technical proofs. In each case the result proved is restated first.

2.2.14. Lemma. A sequence $s$ is strongly fair iff every alternating subsequence is strongly fair.

Proof. Clearly if each alternating subsequence of $s$ is strongly fair, then each such subsequence is also fair and, thus, $s$ is strongly fair. If $s$ is strongly fair then each alternating subsequence $s'$ of $s$ is fair. Since every alternating subsequence of an alternating subsequence is also an alternating subsequence of the original sequence, it follows that each alternating subsequence of $s'$ is fair and, thus, $s'$ is strongly fair. □

2.2.15. Lemma. Let $s$ be strongly fair and $s' = \text{sub}(\pi, s)$ be the alternating subsequence of $s$ at position $\pi$. Let $s'$ contain only a finite number of $\bot$ values. Then for every total $t$ such that $t \geq_p s$, $\text{sub}(\pi, t)$ is fair.

Proof. Since $s'$ contains only a finite number of $\bot$ values, it has a total tail and since $s'$ is fair, that total tail must be fair. Therefore an arbitrary change to the $\bot$ values of $s'$ will result in a fair sequence. So an arbitrary change to the $\bot$ values of $s$ will result in a fair sequence at $\pi$.

2.2.16. Lemma. If $s$ is strongly fair, then there is a strongly fair total $t \geq_p s$.

Proof. For every position $\pi$ such that $\text{sub}(\pi, s)$ contains only a finite number of $\bot$ values, and for every $t \geq_p s$, $\text{sup}(\pi, t)$ will be fair, from Lemma 2.2.15. Hence, we consider only subsequences of $s$ containing an infinity of $\bot$ values. We give now an algorithm for constructing $t$. The algorithm examines each index position $i$ of $s$ in turn and defines $t_i$.

Write $(E_j)$ for some enumeration of the subsequences under consideration such that each one occurs infinitely often in $(E_j)$. Such an enumeration is possible since a countable union of countable sets is countable. In step $j$ of the algorithm we ensure that both a 0 and a 1 are placed in the subsequence of $t$ specified by $E_j$. Step $j$ may be described as follows. We increase $i$, setting $t_i$ to $s_i$ if $s_i$ is defined and to an arbitrary defined value if not, until $i$ is in the alternating subsequence $E_j$ and $s_i = \bot$. We then set $t_i = 0$. We now increase $i$ again in the same way and set $t_i = 1$.

By construction, $t$ is total and $\geq_p s$. Since each relevant subsequence is served by a step of the algorithm infinitely often, each such subsequence is fair. Thus, $t$ is strongly fair. □

2.3.8. Lemma. The relation $\leq_{AS}$ is reflexive and antisymmetric.
Proof. (1) Reflexivity: For all $s,t,u,s\leq_{\text{AS}} s$ using the trivial update with domain as the empty set.

(2) Antisymmetry: Let $s\leq_{\text{AS}} t$ and $t\leq_{\text{AS}} s$. Write $t=[\sigma_1]s$ and $s=[\sigma_2]t$. We show that $\text{dom } \sigma_1$ is the empty set and, thus, that $t=s$.

Consider an arbitrary index $i$ such that $i$ appears at a position in $\text{dom } \sigma_1$. Then $s_i = 1$ and $t_i$ is defined. Thus, $i$ cannot appear at a position in $\text{dom } \sigma_2$. But then $s_i = t_i$: a contradiction. Thus, no index appears at a position in $\text{dom } \sigma_1$. Therefore, $\text{dom } \sigma_1$ is empty. □

2.3.9. Lemma. The relation $\leq_{\text{AS}}$ is transitive.

Proof. Suppose $s\leq_{\text{AS}} t \leq_{\text{AS}} u$. We shall exhibit a parallel update $pu(\sigma_1, \sigma_2)$ of $s$ which results in $u$. Write $t=[\sigma_1]s$ and $u=[\sigma_2]t$. First we show that $\text{dom } \sigma_1$ and $\text{dom } \sigma_2$ are disjoint and $\text{dom } \sigma_1 \cup \text{dom } \sigma_2$ is independent. To prove disjointness, first let $\pi \in \text{dom } \sigma_1$. Then $\text{sub}(\pi, t)$ is total. Thus, $\text{sub}(\pi, t) \neq \bot^\omega$ and $\pi \notin \text{dom } \sigma_2$. Also if $\pi \in \text{dom } \sigma_2$, $\text{sub}(\pi, t) = \bot^\omega$ and, thus, $\pi \notin \text{dom } \sigma_1$. To prove independence of $\text{dom } \sigma_1 \cup \text{dom } \sigma_2$, it is enough to show that for all $\pi \in \text{dom } \sigma_1$ and for all extensions $\pi'$ of $\pi$ and for all prefixes $\pi''$ of $\pi, \pi', \pi'' \notin \text{dom } \sigma_2$. For that, it is enough to show $\text{sub}(\pi', t) \neq \bot^\omega$ and $\text{sub}(\pi'', t) \neq \bot^\omega$. That is evident since $\text{sub}(\pi, t)$ is total; this completes the proof of independence.

Now we may define the parallel update $pu(\sigma_1, \sigma_2)$ with domain $\text{dom } \sigma_1 \cup \text{dom } \sigma_2$ by

$$
\sigma(\pi) = \sigma_1(\pi) \quad \text{if } \pi \in \text{dom } \sigma_1,
$$

$$
\sigma(\pi) = \sigma_2(\pi) \quad \text{if } \pi \in \text{dom } \sigma_2.
$$

It is clear that $u=[\sigma_2][\sigma_1]s=[pu(\sigma_1, \sigma_2)]s$. It remains to show that for all $\pi \in \text{dom } pu(\sigma_1, \sigma_2)$, $\text{sub}(\pi, s) = \bot^\omega$ and $\text{sub}(\pi, u) \in A$.

(i) If $\pi \in \text{dom } \sigma_1$, then $\text{sub}(\pi, s) = \bot^\omega$. Further, $\text{sub}(\pi, t) \in A$ and, therefore, $\text{sub}(\pi, u) \in A$.

(ii) If $\pi \in \text{dom } \sigma_2$, then $\text{sub}(\pi, u) \in A$ and $\text{sub}(\pi, t) = \bot^\omega$. Further, $\pi$ is independent of all positions in $\text{dom } \sigma_1$ and so $\text{sub}(\pi, s) = \text{sub}(\pi, t) = \bot^\omega$. □

2.3.10. Lemma. Every chain of elements in $(\Omega_{\text{AS}}, \leq_{\text{AS}})$ has an upper bound in $(\Omega_{\text{AS}}, \leq_{\text{AS}})$.

Proof. Consider a chain $\{s_i\}$ with $s_i \leq_{\text{AS}} s_{i+1}$ for all $i \geq 1$. For each $i$, write $s_{i+1} = [\sigma_i]s_i$. First we show that for all $i \neq j$, $\text{dom } \sigma_i$ and $\text{dom } \sigma_j$ are disjoint and $\text{dom } \sigma_i \cup \text{dom } \sigma_j$ is independent. We establish this by showing for all $k$, by induction on $k$, that

(a) for all $i,j$ such that $1 \leq i < j \leq k$, $\text{dom } \sigma_i$ and $\text{dom } \sigma_j$ are disjoint,

(b) $\bigcup_{i=1}^k \text{dom } \sigma_i$ is independent.

(c) there is a parallel update $\sigma_{1,k}$ with domain $\bigcup_{i=1}^k \text{dom } \sigma_i$ such that $s_{k+1} = [\sigma_{1,k}]s_1$ and for all $\pi \in \text{dom } \sigma_{1,k}$, $\text{sub}(\pi, s_1) = \bot^\omega$ and $\text{sub}(\pi, s_{k+1}) \in A$. 
For $k = 2$, the hypothesis is established by the argument of Lemma 2.3.9. Assuming the hypothesis for $k = n$, we have that $s_{n+1} = [\sigma_1, \ldots, \sigma_n] s_1$ and $s_{n+2} = [\sigma_{n+1}] s_{n+1}$. Now the argument of Lemma 2.3.9 shows that $\text{dom} \sigma_1, \ldots, \sigma_n$ and $\text{dom} \sigma_{n+1}$ are disjoint and independent. Thus, for all $1 \leq i \leq n$, $\text{dom} \sigma_{n+1}$ and $\text{dom} \sigma_n$ are disjoint and independent. Therefore, $\bigcup_{i=1}^{n+1} \text{dom} \sigma_i$ is independent. Now $\sigma_{1, n+1}$ may be defined by

$$
\sigma_{1, n+1}(\pi) = \sigma_{1, n}(\pi) \quad \text{if} \ \pi \in \text{dom} \sigma_{1, n},
$$

and the argument of Lemma 2.3.9 may be applied again to complete the verification of the hypothesis for $k = n + 1$. Next we define the parallel update $\sigma$ with domain $\bigcup_{i=1}^{\infty} \text{dom} \sigma_i$ by

$$
\sigma(\pi) = \sigma_i(\pi) \quad \text{if} \ \pi \in \text{dom} \sigma_i.
$$

Now define $s = [\sigma] s_1$. We verify that $s$ is an upper bound of $\{s_i\}$. First define the updates $\sigma_j$ with domain $\bigcup_{i \geq j} \text{dom} \sigma_i$ by

$$
\sigma_j(\pi) = \sigma_i(\pi) \quad \text{if} \ i \geq j \ \text{and} \ \pi \in \text{dom} \sigma_i.
$$

It is easy to verify that for all $j \geq 2$, $\sigma = \sigma_j \circ \sigma_{j-1}$ and thus $s = [\sigma] s_1 = [\sigma_j] s_j$. Further, it is clear that for all $\pi \in \text{dom} \sigma_j$, $\text{sub}(\pi, s_j) = 1^\omega$ and $\sigma_j(\pi) \in A$. It follows that for all $j, s \geq s_j$ and, thus, $s$ is an upper bound of $\{s_i\}$.

Finally, we must verify that $s$ is strongly fair. Consider each alternating subsequence of $s$, i.e. consider an arbitrary position $\pi \in P$ and $\text{sub}(\pi, s)$. We must verify that $\text{sub}(\pi, s)$ is fair. There are three cases:

1. $\pi$ is neither a prefix nor an extension of any position in $\text{dom} \sigma$. In this case $\text{sub}(\pi, s) = \text{sub}(\pi, s_1)$ and so is fair (indeed, strongly fair).
2. $\pi$ is an extension of a $\pi'$ in $\text{dom} \sigma$. As $\text{sub}(\pi', s)$ is total and strongly fair, $\text{sub}(\pi, s)$ is fair (indeed, total and strongly fair).
3. $\pi$ is a prefix of a $\pi'$ in $\text{dom} \sigma$. As $\text{sub}(\pi', s)$ is a total and strongly fair subsequence of $\text{sub}(\pi, s)$, $\text{sub}(\pi, s)$ is fair.

2.3.12. Lemma. If $\sigma, \sigma'$ are parallel updates with the same domain $\Pi$, if $\delta, \delta' \in \Omega$ and if $\sigma \leq_{AS} \sigma'$ and $\delta \leq_{AS} \delta'$, then

$$
[\sigma] \delta \leq_{AS} [\sigma'] \delta'.
$$

Proof. We write $\delta' = [\sigma_i] \delta$, such that for all $\pi \in \text{dom} \sigma_i$, $\text{sub}(\pi, \delta) = 1^\omega$ and $\text{sub}(\pi, \delta') \in A$. For every $\pi \in \Pi$, $\sigma(\pi) \leq_{AS} \sigma'(\pi)$, so we write $\sigma'(\pi) = [\sigma_\pi] \sigma(\pi)$, where for all $\pi' \in \text{dom} \sigma_\pi$, $\text{sub}(\pi', \sigma(\pi)) = 1^\omega$ and $\text{sub}(\pi', \sigma'(\pi)) \in A$. We must show the existence of a $\sigma_2$ such that $[\sigma'] \delta' = [\sigma_2] \delta$ and for all $\pi' \in \text{dom} \sigma_2$, $\text{sub}(\pi', [\sigma] \delta) = 1^\omega$ and $\text{sub}(\pi', [\sigma'] \delta') \in A$. We construct $\sigma_2$ as follows. We define

$$
\text{Indep}(\Pi) = \{ \pi \in P \mid \pi \in \text{dom} \sigma_i \},
$$

$$
\text{Sub}(\Pi) = \{ \pi \in P \mid \pi = \pi'. \pi'' \}, \text{ where } \pi' \in \Pi, \pi'' \in \text{dom} \sigma_i\}.
$$
Clearly $\text{Indep}(II)$ and $\text{Sub}(II)$ are disjoint and their union is independent. Now define $\sigma_2$ by
\[
\sigma_2(\pi) = \begin{cases} 
\sigma_1(\pi) & \text{if } \pi \in \text{Indep}(II), \\
\sigma_{\pi}(\pi'') & \text{if } \pi \in \text{Sub}(II), \text{ where } \pi = \pi'.\pi''.
\end{cases}
\]
First we verify that for all $\pi \in \text{dom } \sigma_2$, $\text{sub}(\pi, [\sigma ]\delta) = \perp^\omega$ and $\text{sub}(\pi, [\sigma' ]\delta') \in \Delta$.

(i) If $\pi \in \text{Indep}(II)$, then $\text{sub}(\pi, \delta) = \perp^\omega$ and $\text{sub}(\pi, [\sigma ]\delta) \in \Delta$. Since $\pi$ is independent of all positions in $II$,
\[
\text{sub}(\pi, [\sigma ]\delta) = \text{sub}(\pi, \delta) = \perp^\omega
\]
and
\[
\text{sub}(\pi, [\sigma' ]\delta') = \text{sub}(\pi, \delta') = \text{sub}(\pi, [\sigma ]\delta) \in \Delta.
\]
(ii) If $\pi \in \text{Sub}(II)$, we write $\pi = \pi'.\pi''$, where $\pi' \in II$, $\pi'' \in \text{dom } \sigma_x$. Then
\[
\text{sub}(\pi, [\sigma ]\delta) = \text{sub}(\pi'', \text{sub}(\pi', [\sigma ]\delta)) = \text{sub}(\pi'', \sigma(\pi)) = \perp^\omega.
\]
Further,
\[
\text{sub}(\pi, [\sigma' ]\delta') = \text{sub}(\pi'', \text{sub}(\pi', [\sigma' ]\delta')) = \text{sub}(\pi'', \sigma'(\pi)) \in \Delta.
\]
Finally, we show that $[\sigma' ]\delta' = [\sigma_2 ][\sigma ]\delta$ by showing for each $i$ that the $i$th terms are equal.

Case A: $i$ appears at some $\pi \in II$. Then, $([\sigma' ]\delta')_i = ([\sigma'] [\sigma_1 ]\delta)_i = \sigma'(\pi)_n$ for $n$ as in Definition 2.2.12, and $([\sigma_2 ][\sigma ]\delta)_i = \text{sub}(\pi, [\sigma_2 ][\sigma ]\delta)_n$. It is straightforward to verify that for arbitrary $\delta^*$, $\text{sub}(\pi, [\sigma_2 ]\delta^*) = [\sigma_x ]\text{sub}(\pi, \delta^*)$. Thus,
\[
\text{sub}(\pi, [\sigma_2 ][\sigma ]\delta) = [\sigma_x ]\text{sub}(\pi, [\sigma ]\delta) = [\sigma_x ]\sigma(\pi) = \sigma'(\pi).
\]
Hence, $([\sigma_2 ][\sigma ]\delta)_i = \sigma'(\pi)_n$ as required.

Case B: $i$ does not appear at any position in $II$. There are two possibilities.

(a) $i$ appears at $\pi \in \text{dom } \sigma_1$. In this case $\pi \in \text{Indep}(II)$ and $([\sigma_2 ])[\sigma ]\delta)_i = \text{sub}(\pi, [\sigma_2 ][\sigma ]\delta)_n$ for $n$ as in 2.2.12.

Thus,
\[
\text{sub}(\pi, [\sigma_2 ][\sigma ]\delta)_n = \text{sub}(\pi, [\sigma_1 ]\delta)_n
\]
and
\[
([\sigma' ][\sigma_1 ][\sigma ]\delta)_i = ([\sigma' ][\sigma_1 ]\delta)_i.
\]
(b) $i$ does not appear at any position in $\text{dom } \sigma_1$. In this case $i \notin \text{dom } \sigma_2$. Thus,
\[
([\sigma' ][\sigma_1 ]\delta)_i = ([\sigma_2 ][\sigma ]\delta)_i = \delta_i.
\]

2.3.13. Lemma. Let $\delta$ be total and strongly fair, and let $\sigma$ be an update such that $\text{dom } \sigma$ is finite and for all $\pi \in \text{dom } \sigma$, $\sigma(\pi) = \perp^\omega$. Then, $[\sigma ]\delta \geq_{AS} \perp^\omega$. 

Proof. We define an update $\sigma'$ such that $[\sigma'][\bot^\omega]=[\sigma]\delta$ and for all $\pi \in \text{dom } \sigma', \text{sub}(\pi, [\sigma'] \downarrow^\omega) = \bot^\omega$ and $\text{sub}(\pi, [\sigma'] \downarrow^\omega) \in A$. First we define the domain $M\text{Indep}$ of $\sigma'$ as follows.

$$\text{Indep} = \{ \pi \in P \mid \pi \text{ is independent of every position in } \text{dom } \sigma \}$$

$$M\text{Indep} = \{ \pi \in \text{Indep} \mid \text{there does not exist a proper prefix } \pi' \text{ of } \pi \text{ such that } \pi' \in \text{Indep} \}.$$

Note that if $\pi \in \text{Indep}$, there must be a prefix $\pi'$ of $\pi$ such that $\pi' \in M\text{Indep}$, since all positions are of finite length. $M\text{Indep}$ is clearly independent. Now define $\sigma': M\text{Indep} \rightarrow \Omega$ by

$$\sigma'(\pi) = \text{sub}(\pi, \delta) \quad \text{for all } \pi \in M\text{Indep}.$$ 

Clearly, for all $\pi \in \text{dom } \sigma'$, $\text{sub}(\pi, [\sigma'] \downarrow^\omega) = [\sigma]\delta$. Consider an arbitrary index $i$. There are two cases.

Case A: $i$ does not appear at any position in $\text{dom } \sigma$, so that $\delta_i = ([\sigma]\delta)_i$. Now, $i$ cannot appear at any position which is an extension of a position $\pi$ in $\text{dom } \sigma$, for then $i$ would appear at $\pi$. However, $i$ appears at positions of arbitrarily large length and since $\text{dom } \sigma$ is finite, $i$ must appear at a position whose length is greater than the lengths of all positions in $\text{dom } \sigma$. Thus, $i$ appears at some position in $\text{Indep}$ at some position $\pi \in M\text{Indep}$. For all such $\pi'$, $\text{sub}(\pi', [\sigma'] \downarrow^\omega) = \text{sub}(\pi', \delta)$ and therefore $([\sigma'] \downarrow^\omega)_i = \delta_i - ([\sigma]\delta)_i$.

Case B: $i$ appears at some position in $\text{dom } \sigma$. Thus $([\sigma]\delta)_i = \bot$. Further, $i$ cannot appear at any position in $\text{dom } \sigma'$ and so $([\sigma'] \downarrow^\omega)_i = 1$.

4.5.1. Theorem (associativity of deterministic linking). An arbitrary number of parallel links is equivalent to a series of step-by-step linkings.

Proof. The following proof is due to Staples and Paterson (personal communication).

For a proof by induction it is enough to show that the computed behaviours of the networks constructed by the two different methods indicated in Fig. 8 are equal. Formally, we show that $(X_0, Y_0) = (X_1, Y_1)$, where

$$X_0 = f(X_0, Y_0), \quad Y_0 = g(X_0, Y_0), \quad (1)$$

$$X_1 = a(Y_1), \quad Y_1 = g(a(Y_1), Y_1),$$

$$a(Y) = f(a(Y), Y), \quad (2)$$

with

$$f: D_0 \times D_1 \rightarrow D_0, \quad g: D_0 \times D_1 \rightarrow D_1.$$

First we show that $X_1, Y_1$ is a fixed point of (1). Now

$$X_1 = a(Y_1) = f(a(Y_1), Y_1) \quad \text{and} \quad Y_1 = g(a(Y_1), Y_1).$$
As \((X_0, Y_0)\) is the least fixed point of (1), to show \((X_1, Y_1) = (X_0, Y_0)\) it is enough to show that \(X_1 \leq X_0\) and \(Y_1 \leq Y_0\). To show that \(Y_1 \leq Y_0\) it is enough, by Park’s Rule (cf. [9]), to show that \(g(a(Y_0), Y_0) \leq Y_0\). For this note that \(a(Y_0) = pX.f(X, Y_0)\). But \(\mu X.f(X, Y_0) \leq X_0\) since \(X_0\) is a solution of \(X = f(X, Y_0)\). So \(g(a(Y_0), Y_0) \leq g(X_0, Y_0) = Y_0\), as required. Finally, it is now easy to check that \(X_1 = \mu X.f(X, Y_1) \leq \mu X.f(X, Y_0) \leq X_0\), as required.

5.4.3. Lemma. If \(F, G\) have compensation structures, then so does \(FG\).

Proof. Let

\[ G \in \left[ t_1 \rightarrow s_1 \right] \times \cdots \times \left[ t_n \rightarrow s_n \right] \ni \left[ r_1 \rightarrow s_1 \right] \rightarrow \left[ r_1 \rightarrow s_1 \right] \times \cdots \times \left[ r_n \rightarrow s_n \right] \]

with compensation structure \(\{ (I_{G,j}, W_{G,j}, I_{G,j}) : j = 1, \ldots, n \}\).

Let

\[ F \in \left[ r_1 \rightarrow s_1 \right] \times \cdots \times \left[ r_n \rightarrow s_n \right] \ni \left[ t_1 \rightarrow u_1 \right] \times \cdots \times \left[ t_p \rightarrow u_p \right] \]
with compensation structure \( \{ (\Pi_{F,j}, w_{F,j}, I_{F,j}) : j = 1, \ldots, p \} \).

Consider the \( j \)th component of \( K = FG, j = 1, \ldots, p \). Note that

\[
K_j(f_1, \ldots, f_m) = F_j(G_1(f_1, \ldots, f_m), \ldots, G_n(f_1, \ldots, f_m)).
\]

We write \( f = (f_1, \ldots, f_m) \). We exhibit the required functions for \( K \). We write e.g. \( v_{F,j} \) for the inverse of \( w_{F,j} \), i.e. \( v_{F,j}(h) \) denotes the set of positions which \( w_{F,j} \) maps to \( h \).

1. For \( h = 1, \ldots, m \), \( v_{K,j}(h) = \bigcup_{k=1}^n \text{concat}(v_{F,j}(k), v_{G,k}(h)) \).
2. \( \Pi_{K,j} = \bigcup_{n=1}^m v_{K,j}(h) \).
3. To define \( I_{K,j} : \text{dom } G \times \Pi_{K,j} \times t_j \times \Delta \to \bigcup_{i=1}^m p_i \), note that each \( \pi \in \Pi_{K,j} \) has a unique decomposition \( \pi = \pi_1, \pi_2, \pi_1 \in v_{F,j}(k), \pi_2 \in v_{G,k}(h) \). Then we define

\[
I_{K,j}(f_1, \pi, X, \delta) = I_{G,k}(f_1, \pi_2, I_{F,j}((G_1(f)), \ldots, G_n(f)), \pi_1, X, \delta), \text{sub(} \pi_1, \delta)\).
\]

Properties 5.3.2(a)-(e) are straightforward to check. We verify property 5.3.2(f) now. We write \( w \) for \( w_{K,j} \). Let \( \sigma \) be an update from \( \Pi_{K,j} \) to \( \Delta \) such that for all \( \pi \in \Pi_{K,j}, \sigma(\pi) \) compensates \( f_{w(\pi)} \) to \( g_{w(\pi)} \) on \( I_{K,j}(f_1, \pi, X, \delta), \text{sub(} \pi_1, \delta) \). That is, for all \( h = 1, \ldots, m \), all \( k = 1, \ldots, n \), all \( \pi_1 \in v_{F,j}(k) \) and \( \pi_2 \in v_{G,k}(h) \), \( \sigma(\pi_1, \pi_2) \) compensates \( f_h \) to \( g_h \) on

\[
I_{G,k}(f_2, \pi, X, \delta) = I_{G,k}(f_1, \pi_2, I_{F,j}((G_1(f)), \ldots, G_n(f)), \pi_1, X, \delta), \text{sub(} \pi_1, \delta)\).
\]

Now for each \( k = 1, \ldots, n \) such that \( v_{F,j}(k) \) is nonempty, define the update \( \sigma'_k \) from \( \bigcup_{h=1}^m v_{G,k}(h) \) to \( \Delta \) as follows. For all \( \pi_2 \in \bigcup_{h=1}^m v_{G,k}(h), \sigma'_k(\pi_2) = \sigma(\pi_1, \pi_2) \), where \( \pi_1 \) is the unique element of \( v_{F,j}(k) \) such that \( \pi_1, \pi_2 \in \Pi_{K,j} \). Then, for each \( k \) such that \( v_{F,j}(k) \) is nonempty, for all \( \pi_2 \in v_{G,k}(h), h = 1, \ldots, m \), \( \sigma'_k(\pi_2) \) compensates \( f_h \) to \( g_h \) on

\[
I_{G,k}(f_1, \pi_2, I_{F,j}((G_1(f)), \ldots, G_n(f)), \pi_1, X, \delta), \text{sub(} \pi_1, \delta)\).
\]

We now use property 5.3.2(f) applied to \( G_1, \ldots, G_n \). For each \( k = 1, \ldots, n \) and each \( \pi_1 \in v_{F,j}(k) \), \( [\sigma'_k] \text{sub(} \pi_1, \delta) \) compensates \( G_k(f) \) to \( G_k(g) \) on

\[
I_{F,j}((G_1(f)), \ldots, G_n(f)), \pi_1, X, \delta), \text{sub(} \pi_1, \delta)\).
\]

Next we define the update \( \sigma' \) from \( \bigcup_{h=1}^m v_{F,j}(k) \) to \( \Delta \) by

\[
\sigma'(\pi_1) = [\sigma'_k] \text{sub(} \pi_1, \delta) \quad \text{for each } \pi_1 \in v_{F,j}(k)\).
\]

Then using property 5.3.2(f) applied to \( F_j, [\sigma'] \delta \) compensates \( F_j(G_1(f)), \ldots, (G_n(f)) \) to \( F_j(G_1(g)), \ldots, (G_n(g)) \) on \( X, \delta \). Finally, it is easy to check that \( [\sigma'] \delta = [\sigma] \delta \); thus, concluding the proof. \( \square \)

**5.4.8. Lemma.** If \( G \) is modelling and has the infinite history property, then \( \text{L} \text{INK}_G \), with compensation structure as defined in Lemma 5.4.3, has the infinite history property.

**Proof.** Write \( K = \text{L} \text{INK}_G \). The compensation structure defined in Lemma 5.4.3 for \( \text{L} \text{INK}_G \) is as follows. Here, in the notation of Lemma 5.4.3, \( n = 1, p = 1 \) and we omit
these subscripts. We also write \( X = (X_1, \ldots, X_{i-1}, X_i, X_{i+1}, \ldots, X_m) \).

1. For \( h = 1, \ldots, m \), \( v_k(h) = v_G(h) \).

2. \( \Pi_k = \Pi_G \).

3. \( \nu_k(f, \pi, X, \delta) = \nu_G(f, \pi, G(f)(X_1, \ldots, X_{i-1}, H, X_{i+1}, \ldots, X_m, \delta), \delta) \), where, as in the definition of compensation structure for \( \text{LINK}_k \), \( H \) is the least fixed point of

\[
H = [G(f)(X_1, \ldots, X_{i-1}, H, X_{i+1}, \ldots, X_m, \delta)]_k.
\]

Now to show the infinite history property, assume that \( f \) is causal and \( X \) is infinite at all components. Thus, \( fd f \) and \( X d X \). Then the proof of Lemma 4.4.3 shows that \( H d H \) and, hence, that \( H \) is infinite. Now \( G(f) \) is causal, so using the infinite history property of \( G \) we obtain the desired result.

Acknowledgment

The authors thank Professor David Park, Dr. Ian Hayes and the referees for helpful criticism and comments.

References


