Available online at www.sciencedirect.com
ScienceDirect

Electronic Notes in Theoretical Computer Science 175 (2007) 109–129

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Event Structure Spans for Nondeterministic Dataflow

Lucy Saunders-Evans [1]

University of Cambridge Computer Laboratory, England

Glynn Winskel [2]

University of Cambridge Computer Laboratory, England

Abstract

A compositional semantics for nondeterministic dataflow processes is described using spans of event structures; such a span describes a computation between datatypes, themselves represented by event structures, as itself an event structure. The spans of event structures represent certain profunctors (a generalisation of relations) used previously in a compositional semantics of nondeterministic dataflow and in the semantics of higher-order processes. Deterministic spans of event structures are shown to correspond to stable continuous functions and their semantics of dataflow to reduce to the usual least fixed-point semantics of Kahn.

Keywords: Event Structures, Spans, Nondeterministic Dataflow, Stable functions.

1 Introduction

A dataflow process is built as a network of more basic processes connected by channels. In particular processes may be connected to allow feedback loops from output to input.

In [6], Kahn demonstrated that a denotational semantics could be given to a dataflow network of deterministic processes as a least fixed point solution of a set of equations describing the components.

Brock and Ackerman showed in [2] that giving a semantics to nondeterministic dataflow processes was far from easy. In particular, *input-output* relations between sequences of values on input and output channels carry too little information about the behaviour of networks to support a compositional semantics. The following

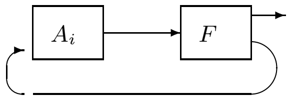
[1] Email: `ls275@cam.ac.uk`
[2] Email: `gw104@cam.ac.uk`

doi:10.1016/j.entcs.2006.10.055

example is essentially that presented in [11]. Let A_1 be the nondeterministic process that either outputs a token and stops, or outputs a token, waits for a token on input and then outputs another token. Let A_2 be the process that either outputs a token and stops, or waits for a token on input and then outputs two tokens. The input-output relations between strings of tokens of both A_1 and A_2 are the same:

$$\{(\varepsilon,\ t),\ (t,\ t),\ (t,\ tt)\}\ ,$$

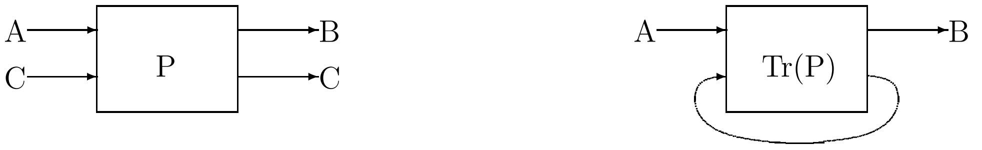
where ε represents the empty string of tokens and the presence of a token is represented by the symbol t. Let F be the process that copies every input to two outputs through which the output of the process A_i is fed back to the input channel. Observe that the process A_1 placed in this context produces a process which can output two tokens whereas the process A_2 results in a process that can only output a single token.

$C[A_i] =$ 

This confirms that there is no denotational semantics of nondeterministic dataflow in terms of traditional input-output relations. For similar reasons traditional uses of powerdomains also fall short.

Although traditional relations are insufficient, it was however shown in [4,5] that a compositional form of relational semantics is possible, but at the cost of moving to generalised relations called profunctors. Here we provide a representation of the *stable port profunctors* used in [4,5] as *spans of event structures*. The intention is to fit nondeterministic dataflow within a versatile theory of processes and types based on a general theory of spans of event structures [14,12].

In more recent years feedback of the kind found in dataflow has reappeared in a variety of new contexts, which are condensed in a more abstract and general formulation of the properties a feedback operation, called *trace*, should satisfy. Let $\mathcal{C}$ be a category with a symmetric monoidal structure, $\otimes$. A trace operation for $\mathcal{C}$ is defined to be a family of operations, $Tr^{C}_{A,B} : \mathcal{C}(A \otimes C,\ B \otimes C) \to \mathcal{C}(A,\ B)$. The intuition behind this operation is illustrated in the following diagram:



A trace operation must obey a number of axioms to ensure it behaves correctly. The reader is referred to [4,5], which this paper supplements, for the detailed axioms and fuller set of references to this rich area.

2 Spans of Event Structures

Event structures model a process as a set of event occurrences with relations to express how events causally depend on others, or exclude other events from occurring.

Definition 2.1 An event structure is a tuple, $(E, \leq, \#)$, where:

- E is a set of events;
- $\leq$ is a partial order on E. Define $[e]$ to be $\{e' \mid e' \leq e\}$. It must be the case that the set $[e]$ is finite for all $e \in E$;
- $\#$ is a binary, irreflexive, symmetric relation such that $e_1 \# e_2$ and $e_2 \leq e_3$ implies that $e_1 \# e_3$.

A *configuration* of an event structure is a downwards closed subset of events in which no two events are in conflict. Define $\mathcal{C}(E)$ to be the set of configurations of E. □

The set of configurations of an event structure when ordered by inclusion form a prime algebraic domain [8]; the complete primes have the form $[e]$ for an event e.

Definition 2.2 A set of configurations X of an event structure E is *compatible*, written $X\uparrow$, iff there is a configuration y such that $\forall x \in X.\ x \subseteq y$; then $\bigcup X$ is itself a configuration. In the case where $X = \{w, z\}$ we usually write $w \uparrow z$ instead of $X\uparrow$. □

Since their introduction in [8] it has been recognised that event structures can represent both processes and domains. At the time, in [8], this was remarked on as creating a 'curious mismatch'; in traditional denotational semantics, a process denotes only an element of a domain. The ambiguous double role of event structures can be resolved by working with spans of event structures. A span

$$\begin{tikzcd} & E \arrow[dl, "in"'] \arrow[dr, "out"] & \\ A & & B \end{tikzcd}$$

represents a process computing from a type, represented by event structure A, to the type B, another event structure, as again itself an event structure E. The morphisms *in* and *out* specify how the event structure E inspects input and delivers output. There are many possible variations on spans as the morphisms *in* and *out* can have different natures. A systematic way to explore the range of possibilities via monads and distributive laws was suggested in [14] and is the subject of ongoing research [12]. For the purposes of this paper however we can restrict to very specific spans in which *in* is a 'demand' morphism and *out* is 'rigid.' Such spans first arose unexpectedly as representations of the profunctors used in a denotational semantics of higher-order processes [9].

A *rigid* morphism, $f : E_1 \to E_2$, between event structures E_1 and E_2 consists of a function between the respective sets of events such that

- $[f(e)] = f[e]$, for all events e in E_1, and
- if $f(e) = f(e')$ or $f(e) \# f(e')$, then $e = e'$ or $e \# e'$, for all e and e' in E_1.

Rigid morphisms compose as functions. In proofs we will use an alternative characterisation of rigid morphisms, and properties based on their effect on configurations.

Proposition 2.3 *Let E_1 and E_2 be event structures. A function f from the set of events E_1 to the set of events E_2 is a rigid morphism $f : E_1 \to E_2$ iff it*

- *preserves causal dependency,*

$$\forall e, e' \in E_1.\ e \leq e' \Rightarrow f(e) \leq f(e')\ ,\ \text{and}$$

- *preserves configurations,*

$$\forall x \in \mathcal{C}(E_1).\ fx \in \mathcal{C}(E_2)\ \&\ \forall e, e' \in x.\ f(e) = f(e') \Rightarrow e = e'\ .$$

A rigid morphism $f : E_1 \to E_2$ determines a stable *function on configurations [1]:*

$$\forall x, y \in \mathcal{C}(E_1).\ x \uparrow y \Rightarrow f(x \cap y) = (f\,x) \cap (f\,y)\ .$$

A *demand* morphism $d : E_1 \to E_2$ is a function from the set of events E_1 to the finite configurations of E_2. It must obey the following conditions:

- $e \leq e'$ implies $d(e) \subseteq d(e')$, for all events $e, e' \in E_1$;
- $d(e) \not\uparrow d(e')$ implies $e \# e'$, for all events $e, e' \in E_1$.

A demand morphism can be extended to act on configurations as well as events. Let $d : E_1 \to E_2$ be a demand morphism between E_1 and E_2. We can extend this to a function $d^\dagger : \mathcal{C}(E_1) \to \mathcal{C}(E_2)$ by

$$d^\dagger(x) \stackrel{def}{=} \bigcup_{e \in x} d(e)$$

for $x \in \mathcal{C}(E_1)$. The fact that $d^\dagger(x)$ is a configuration follows directly from x being a configuration and that $d(e_1) \not\uparrow d(e_2)$ implies that $e_1 \# e_2$. Clearly, if x is a finite configuration, then so is $d^\dagger(x)$, and the extension $d^\dagger$ sends finite configurations to finite configurations. The composition of two demand morphisms $d_2 \circ \mathrm{d}_1$ is achieved by the composition of functions $d_2^\dagger \circ d_1$.

For this paper, we restrict attention to spans of a special kind:

Definition 2.4 A span of event structures comprises $A \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B$ where E, A and B are event structures, d is a demand morphism and *out* is a rigid morphism. (Where there is no confusion, we refer to a span by its vertex, *i.e.*, $A \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B$ will be referred to as E.) □

We can compose spans sequentially. Given two spans $A \overset{d_1}{\leftarrow} E_1 \overset{out_1}{\rightarrow} B$ and $B \overset{d_2}{\leftarrow} E_2 \overset{out_2}{\rightarrow} C$, their composition is the span $A \overset{d}{\leftarrow} E_3 \overset{out}{\rightarrow} C$ where E_3 comprises the event structure with

- *Events* $\{(x,\ e) \in \mathcal{C}(E_1) \times E_2 \mid out_1 x = d_2(e)\}$;
- *Causal dependency* $(x,\ e) \leq (x'\ e')$ iff $x \subseteq x'$ and $e \leq e'$;

- *Conflict* $(x,\ e)\#(x',\ e')$ iff $x \not\uparrow x'$ or $e\#e'$.

The demand and output morphisms of the composition d and out are given by

$$d(x,e) \overset{def}{=} d_1^{\dagger}(x) \text{ and } out(x,e) \overset{def}{=} out_2(e)\ .$$

Because rigid morphisms can be identified with (certain kinds of) demand morphisms, this construction can be viewed as a pullback construction in the category of event structures with demand morphisms. (The spans here form a bicategory in the usual manner of spans [7].)

There is also a parallel composition of spans (which forms the symmetric monoidal structure, $\otimes$, referred to in Section 1).

Definition 2.5 Given two event structures, $(E_1,\ \leq_1,\ \#_1)$ and $(E_2,\ \leq_2,\ \#_2)$, their *parallel composition* $(E_1,\ \leq_1,\ \#_1) \otimes (E_2,\ \leq_2,\ \#_2)$ is the event structure with:

- *Events* $E_1 \uplus E_2$, the disjoint union of events, $(\{1\} \times E_1) \cup (\{2\} \times E_2)$,
- *Causal dependency* $(i,\ e_1) \leq (j,\ e_2)$ iff $i = j$ and $e_1 \leq_i e_2$, and
- *Conflict* $(i,\ e_1)\#(j,\ e_2)$ iff $i = j$ and $e_1\#_i e_2$.

□

Once the event structures are made disjoint their parallel composition is given by the union of their events and relations of causality and conflict.

The parallel composition of $A \overset{d_1}{\leftarrow} E_1 \overset{out_1}{\rightarrow} B$ and $C \overset{d_2}{\leftarrow} E_2 \overset{out_2}{\rightarrow} D$ is

$$\begin{tikzcd} & E_1 \otimes E_2 \arrow[dl, "d_1 \otimes d_2"'] \arrow[dr, "out_1 \otimes out_2"] & \\ A \otimes C & & B \otimes D \end{tikzcd}$$

where $(d_1 \otimes d_2)(i,\ e) \overset{def}{=} \{i\} \times d_i(e)$ and $(out_1 \otimes out_2)(i,\ e) \overset{def}{=} (i,\ out_i(e))$ for $i = 1, 2$.

From these constructions, it is possible to give a semantics to nondeterministic dataflow.

3 Stable Families

It is sometimes difficult to define constructions on event structures directly and it is often simpler to first define them in terms of stable families, from which an event structure can then be obtained from the prime configurations. Stable families will be used to define the trace construction.

Definition 3.1 A stable family, $\mathcal{F}$, is a set of sets with the following properties.

- *Coherence* $\forall X \subseteq \mathcal{F}.\ (\forall x,\ y \in X.\ x \uparrow y) \Rightarrow \bigcup X \in \mathcal{F}$.
- *Stability* $X\uparrow\ \Rightarrow \bigcap X \in \mathcal{F}$ for all non-empty $X \subseteq \mathcal{F}$.
- *Coincidence-freeness*

$$\forall x \in \mathcal{F},\ e_1,\ e_2 \in x.\ e_1 \neq e_2 \Rightarrow$$
$$\exists y \in \mathcal{F}.\ y \subseteq x\ \&\ ((e_1 \in y)\ \&\ (e_2 \notin y)) \text{ or } ((e_2 \in y)\ \&\ (e_1 \notin y)).$$

- *Finiteness* $\forall x \in \mathcal{F}, e \in x\, \exists y \in \mathcal{F}.\ y \subseteq x \;\&\; e \in y \;\&\; |y| < \infty$.

$\square$

It can be deduced that a stable family, ordered by inclusion, forms a *prime algebraic* domain [8,13]. If x is a member of a stable family $\mathcal{F}$ and $e \in x$ let

$$[e]_x \overset{def}{=} \bigcap\{y \in \mathcal{F} | e \in y \;\&\; y \subseteq x\}.$$

Then $[e]_x \in \mathcal{F}$ and is called a *prime configuration* of $\mathcal{F}$. (It is a complete prime of $(\mathcal{F}, \subseteq)$ in the sense of [8].) Because of coincidence-freeness, taking $max([e]_x) \overset{def}{=} e$ is well-defined.

4 Trace for Event Structures

This section is devoted to defining a trace operation $Tr^{C}_{A,B}$ which takes a span from $A \otimes C$ to $B \otimes C$ to a span from A to B. Consider a span

$$\begin{tikzcd} & E \arrow[dl, "d"'] \arrow[dr, "out"] & \\ A \otimes C & & B \otimes C \end{tikzcd}.$$

We first build a stable family out of those configurations of E which are *secure*. Roughly a configuration x is secure if the demand of each event $e \in x$ is either met by the input in A, or by previous output to C, which we imagine is fed back as input. This idea is formalised below.

In defining the trace it is convenient to assume that both pairs of sets A, C, and B, C are disjoint, so that we can treat the parallel compositions $A \otimes C$ and $B \otimes C$ as given by union. We can now, for example, write $y \cap C$ for the 'projection' of a configuration $y \in \mathcal{C}(A \otimes C)$ to its component in the event structure C.

Definition 4.1 Let x be a configuration of E. A *securing sequence* in x consists of a sequence of events $e_1, \cdots, e_n$ in x such that

$$\{e_1, \cdots, e_i\} \in \mathcal{C}(E) \;\;\&\;\; d(e_i) \cap C \;\subseteq\; out\{e_1, \cdots, e_{i-1}\}$$

for all $i \leq n$. An event e is *secured* in x iff there is a securing sequence $e_1, \cdots, e_n$ in x with $e_n = e$. Finally, the configuration x is *secure* if each of its events is *secured* in x. $\square$

The subset of $\mathcal{C}(E)$, consisting of all the secure configurations will be shown to be a stable family, from which we then obtain an event structure. Our proofs will make use of a relation $\prec_x$ expressing the extra causal dependency on a configuration x of E which is introduced through feedback.

Definition 4.2 Let x be a configuration of E. For all events e_1 and e_2 in x, define

$e_1 \rightarrow_x e_2$ iff $out(e_1) \in d(e_2) \cap C$, and
$e_1 \prec_x e_2$ iff $e_1 < e_2$ or $e_1 \rightarrow_x e_2$.

We define $\{e\}_x$ to be the set $\{e' \mid e' \prec_x^\star e\}$ for all $e \in x$. (Note that because $\{e\}_x$ must be a downwards closed subset of x with respect to causal dependency it is necessarily a configuration.) □

Lemma 4.3 *An event e is secured in configuration x iff*

(i) *$\prec_x$ is well-founded on $\{e\}_x$, and*

(ii) *$(d^\dagger(\{e\}_x)) \cap C \subseteq out\, x$.*

Proof. *if*: Assume (i) and (ii). To prove that event e is secured in x, we require a securing sequence for e. First note that the set $\{e\}_x$ is finite as $\prec_x$ is well-founded and the set of $\prec_x$-predecessors of an event is also finite. Thus tentatively we may take the securing sequence to be a choice of order $\{e_1, \cdots, e_n\}$ of the set $\{e\}_x$ which respects $\prec_x$, *i.e.*, for $0 < i, j \le n$,

$$(e_i \prec_x e_j \Rightarrow i < j) \text{ and } e_n = e\ .$$

We now check that the chosen sequence is indeed securing. Observe that the set $\{e_1, \cdots,\ e_i\}$, with $i \le n$, is a configuration of E because it is a downwards closed subset of x—this follows immediately from the definition of $\prec_x$. It remains to confirm that $d(e_i) \cap C \subseteq out\{e_1, \cdots,\ e_{i-1}\}$, for all $i \le n$. Consider an event $c \in d(e_i) \cap C$ for some $i \le n$. As $d^\dagger(\{e\}_x) \cap C \subseteq out\, x$, we have that $c = out(e')$ for some $e' \in x$. By definition, $e' \rightarrow_x e_i$, so $e' \prec_x e_i$. Thus $e' \in \{e\}_x$ and, as the sequence respects $\prec_x$, we see that $e' = e_j$ for some $j < i$. This confirms that $c \in out\{e_1, \cdots,\ e_{i-1}\}$. Hence $\{e_1, \cdots, e_n\}$ is a securing sequence.

only if: Assume that $e_1, \cdots,\ e_n = e$ is a securing sequence in x.

We first show that for $i \le n$

(†) $e' \prec_x e_i \Rightarrow e' \in \{e_1, \cdots, e_{i-1}\}$

and therefore that $e' = e_j$ for some $j < i$.

By definition if $e' \prec_x e_i$ then either $e' < e_i$ or $e' \rightarrow_x e_i$. As $\{e_1, \cdots,\ e_i\}$ is a configuration and therefore downwards closed with respect to $<$, if $e' < e_i$ then $e' \in \{e_1, \cdots,\ e_{i-1}\}$. If $e' \rightarrow_x e_i$ then $out(e') \in d(e_i) \cap C$. But $d(e_i) \cap C \subseteq out\{e_1, \cdots, e_{i-1}\}$ by the property of a securing sequence. So $out(e') = out(e_j)$ where $e_j \in \{e_1, \cdots, e_{i-1}\}$. As both $e', e_j \in x$ and out is a rigid morphism $e' = e_j$ so $e' \in \{e_1, \cdots, e_{i-1}\}$.

Now we can show *(i)* and *(ii)*.

(i) By (†) a $\prec_x$-descending chain in $\{e\}_x$ induces a strictly descending chain of finite sets under inclusion—which is certainly well-founded. Hence $\prec_x$ is well-founded on $\{e\}_x$.

(ii) It follows from (†) that $\{e\}_x \subseteq \{e_1, \cdots, e_n\}$, so $d^\dagger(\{e\}_x) \subseteq d^\dagger(\{e_1, \cdots, e_n\})$. Thus

$$\begin{aligned}(d^\dagger(\{e\}_x)) \cap C &\subseteq (d^\dagger(\{e_1, \cdots, e_n\}) \cap C \\ &\subseteq (\bigcup_{i \leq n} d(e_i)) \cap C \\ &\subseteq \bigcup_{i \leq n} (d(e_i) \cap C) \\ &\subseteq \bigcup_{i \leq n} out\{e_1, \cdots, e_{i-1}\} \quad \text{as } e_1, \ldots, e_n \text{ is a securing sequence,} \\ &\subseteq out\, x \ .\end{aligned}$$

□

Corollary 4.4 *A configuration x is secure iff*

(i) $\prec_x$ is well-founded on x, and

(ii) $d^\dagger(x) \cap C \subseteq out\, x$.

Proof. *if*: Assume that a configuration x satisfies (i) and (ii) above. Let $e \in x$. Then certainly $\prec_x$ is well-founded on $\{e\}_x$ and $(d^\dagger(\{e\}_x)) \cap C \subseteq d^\dagger(x) \cap C \subseteq out\, x$. Thus e is secured in x by Lemma 4.3. But e was an arbitrary event in x. Hence x is secure.

only if: Assume the configuration x is secure. By Lemma 4.3, $\prec_x$ is well-founded on x, *i.e.* (i) . To show (ii):

$$\begin{aligned}d^\dagger(x) \cap C &= (\bigcap_{e \in x} d^\dagger(\{e\}_x)) \cap C \\ &= \bigcap_{e \in x} (d^\dagger(\{e\}_x) \cap C) \subseteq out\, x \ , \qquad \text{by Lemma 4.3.}\end{aligned}$$

□

In proving that the family of secure configurations forms a stable family, we will make use of the following lemma.

Lemma 4.5 *Suppose x and y are secure configurations of E with $x \uparrow y$. Let $e \in x \cap y$. Then,*

(i) $e' \prec_x e$ iff $e' \prec_y e$, for all events e', and

(ii) $\{e\}_x = \{e\}_y$.

If x is a secure configuration and $e \in x$, then $\{e\}_x$ is the least secure sub-configuration of x containing e.

Proof. (i) Assume $e \in x \cap y$ where $x \uparrow y$. Suppose $e' \prec_x e$. Then either $e' < e$ or $e' \rightarrow_x e$. In the former case, $e \in y$ and $e' \prec_y e$. In the latter case, $out(e') \in d(e) \cap C$ with $e' \in x$. As y is secure,

$$d(e) \cap C \subseteq (d^\dagger(y)) \cap C \subseteq out\, y \ .$$

Thus $out(e') = out(e'')$ with $e' \in x$ and some $e'' \in y$. But $x \uparrow y$ and out is a rigid morphism, so $e' = e''$ making $e' \in y$. It follows that $e' \rightarrow_y e$, and $e' \prec_y e$. In either case, $e' \prec_x e$ implies $e' \prec_y e$. The same argument proves the converse implication, establishing (i).

(ii) Obvious from (i).

Assume $e \in x$ where x is a secure configuration. From Corollary 4.4, to establish that $\{e\}_x$ is secure it suffices to check (i) that $\prec_{\{e\}_x}$ is well-founded on $\{e\}_x$ and (ii) that $(d^\dagger(\{e\}_x)) \cap C \subseteq out\,\{e\}_x$. Now (i) follows directly because $\prec_x$, which includes $\prec_{\{e\}_x}$, is well-founded on $\{e\}_x$ by Lemma 4.3. We now concentrate on showing (ii). By Lemma 4.3, as x is secure, we have that $(d^\dagger(\{e\}_x)) \cap C \subseteq out\,x$. Thus any event of $(d^\dagger(\{e\}_x)) \cap C$ takes the form $out(e')$ for some $e' \in x$. But supposing $out(e') \in (d^\dagger(\{e\}_x)) \cap C$, then $out(e') \in (d(e'')) \cap C$, *i.e.* $e' \rightarrow_x e''$, for some $e'' \in \{e\}_x$. Consequently, $e' \in \{e\}_x$. Hence $(d^\dagger(\{e\}_x)) \cap C \subseteq out\,\{e\}_x$.

Suppose $e \in y \subseteq x$ where y is a secure configuration. Then certainly $x \uparrow y$, so $\{e\}_x = \{e\}_y \subseteq y$. Thus, $\{e\}_x$ is the least secure sub-configuration of x containing e. □

Theorem 4.6 *The family consisting of all secure configurations of E is a stable family. For any $e \in x$, a secure configuration, $[e]_x = \{e\}_x$.*

Proof. Let $\mathcal{S} = \{x \in \mathcal{C}(E) \mid x \text{ is secure}\}$. We show $\mathcal{S}$ is a stable family.

Coherence: $\forall X \subseteq \mathcal{S}.\ (\forall x,\ y \in X.\ x \uparrow y) \Rightarrow \bigcup X \in \mathcal{S}$.
Assume X is a pairwise compatible subset of $\mathcal{S}$. It is clear that $\bigcup X$ is a configuration of E. If $e \in \bigcup X$ then $e \in x$ for some x in X. As e is secured in x and $x \subseteq \bigcup X$, there is a securing sequence for e in $\bigcup X$.

Stability: $\forall X \subseteq \mathcal{S}.\ X\uparrow \Rightarrow\ \bigcap X \in \mathcal{S}$.
Suppose $X \subseteq \mathcal{S}$ and $X\uparrow$. Then, $\forall x \in X.\ x \subseteq y$ for some secure configuration y. Consider the set

$$Y \overset{def}{=} \{\{e\}_y \mid e \in \bigcap X\ \}\ .$$

It consists of secure configurations which are certainly pairwise compatible, with upper bound y. By coherence, $\bigcup Y$ is a secure configuration. Clearly, $\bigcap X \subseteq \bigcup Y$ as each $e \in \bigcap X$ is a member of $\{e\}_y$. For any $e \in \bigcap X$ and $x \in X$, then $\{e\}_y = \{e\}_x \subseteq x$, by Lemma 4.5. So we also have the reverse inclusion $\bigcup Y \subseteq \bigcap X$ ensuring the equality $\bigcap X = \bigcup Y$, and that $\bigcap X$ is a secure configuration.

Finiteness: $\forall x \in \mathcal{S} \forall e \in x \exists y \in \mathcal{S}.\ y \subseteq x\ \ \&\ e \in y\ \&\ |y| < \infty$.
If x is secure then each event e in x must have a securing sequence. This determines a finite secure configuration in $\mathcal{S}$ which contains e.

Coincidence-freeness: $\forall x \in \mathcal{S},\ e_1,\ e_2 \in x.\ e_1 \neq e_2 \Rightarrow$
$\exists y \in \mathcal{S}.\ y \subseteq x\ \&\ ((e_1 \in y)\ \&\ (e_2 \notin y))$ or $((e_2 \in y)$ and $(e_1 \notin y))$.

Assume $e_1, e_2 \in x \in \mathcal{S}$ and $e_1 \neq e_2$. Consider the secure configurations $\{e_1\}_x$ and $\{e_2\}_x$. If e_2 is a member of $\{e_1\}_x$ then $e_2 \prec_x^+ e_1$. As x is secure, it cannot be the case that $e_1 \prec_x^+ e_2$—otherwise we would contradict the well-foundedness of $\prec_x$. Therefore $e_1 \notin \{e_2\}_x$ if $e_2 \in \{e_1\}_x$ and *vice versa*.

Finally, let $e \in x$, a secure configuration. Both $[e]_x$ (by definition) and $\{e\}_x$ (by Lemma 4.5) are the smallest secure sub-configurations of x which contain e, and hence equal. $\square$

We can now define the trace operation.

Definition 4.7 Define $Tr(A \otimes C \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B \otimes C)$ to be $A \overset{d'}{\leftarrow} P \overset{out'}{\rightarrow} B$ where P is the event structure comprising

- *Events*, the prime configurations p of $\mathcal{S}$ for which $out(max(p)) \in B$, with
- *Causal dependency*, $p_1 \leq p_2$ iff $p_1 \subseteq p_2$, and
- *Conflict*, $p_1 \# p_2$ iff $p_1 \not\uparrow p_2$ in $\mathcal{S}$,

and where, for $p \in P$, we define $d'(p) \overset{def}{=} d^\dagger(p) \cap A$ and $out'(p) \overset{def}{=} out(max(p))$. $\square$

We should check that the definition of trace does indeed yield a span. In order to show that out' is rigid we observe the following.

Lemma 4.8 *For e_1 and e_2 in $x \in \mathcal{C}(E)$, if $e_1 {\prec_x}^\star e_2$ and $e_1 \not\leq e_2$ then $out(e_1) \in C$.*

Proof. By a simple induction on the length of chain $e_1 \prec_x \cdots \prec_x e_2$.
Basis: Suppose that $e_1 \prec_x e_2$. In this case $e_1 \not\leq e_2$ implies that $e_1 \rightarrow_x e_2$, so $out(e_1) \in C$.
Induction step: Suppose that the property holds for all chains of length less than or equal to k, *i.e.*, assume that if $e_1 {\prec_x}^k e_2$ and $e_1 \not\leq e_2$ then $out(e_1) \in C$. Suppose $e_1 {\prec_x}^{k+1} e_2$ and $e_1 \not\leq e_2$. Then there exists an e' with $e_1 {\prec_x}^k e'$ and $e' \prec_x e_2$. If $e_1 \not\leq e'$ then, by the induction hypothesis, $out(e_1) \in C$. If $e_1 \leq e'$ then, as $e_1 \not\leq e_2$, we have $e' \not\leq e_2$ and so $out(e') \in C$. But then $out(e_1) \in C$ as out is a rigid morphism. $\square$

Lemma 4.9 *The map out' is a rigid morphism:*

Proof. We first show that out' preserves causal dependency. Suppose $p' \leq p$ in P. Write $max(p') = e'$ and $max(p) = e$. Then, $e' \prec_p e$ as $[e]_p = \{e\}_p$. Now, by Lemma 4.8, $e' \leq e$ as $out(e') \in B$. Thus $out(e') \leq out(e)$, as out is rigid, which directly yields $out'(p') \leq out'(p)$.

Now let X be a configuration of P. First observe that

$$out'\, X = B \cap out \bigcup X \ .$$

Clearly we have the inclusion $out'\, X \subseteq B \cap out \bigcup X$. To show the reverse inclusion, suppose $e \in \bigcup X$ and $out(e) \in B$. Then, $e \in p$ for some $p \in X$. Write $p' \overset{def}{=} [e]_p$. Because $p' \leq p$ and X is downwards closed, $p' \in X$, with $out'(p') = out(e)$—as required.

It follows from the observation that $out' X$ is a configuration of B: because X is a set of pairwise compatible configurations of E so is their union $\bigcup X$, and therefore so is the image under the morphism *out* and 'projection' to B.

Suppose $p, p' \in X$ and $out'(p) = out'(p')$. The union $\bigcup X$ is a secure configuration, compatible with both p and p'. Thus $p = [e]_{\bigcup X}$ and $p' = [e']_{\bigcup X}$ for some $e, e' \in \bigcup X$. From the definition of out' it follows that $out(e) = out(e')$. Because *out* is a rigid morphism sending both $e, e' \in \bigcup X$ to common event we must have $e = e'$. Hence $p = p'$. □

Theorem 4.10 *The trace* $Tr(A \otimes C \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B \otimes C)$ *is a span.*

Proof. In the definition of the trace $Tr(E)$ as the span $A \overset{d'}{\leftarrow} P \overset{out'}{\rightarrow} B$ it is easily seen that P is an event structure. Lemma 4.9 confirms that out' is indeed a rigid morphism. That d' is a demand morphism follows directly from its definition. □

Relying on previous work [4,5], we can show that the operation Tr obeys the trace axioms up to isomorphism—see Appendix. In [4,5], a trace operation was given for *stable port profunctors*. Spans of event structures represent such profunctors. The representation respects both sequential and parallel composition as well as the trace operation.

5 Deterministic Dataflow

To connect with Kahn's classic treatment of deterministic dataflow we define the notion of a *deterministic span*. Such spans represent stable functions between domains of configurations, and can be used to model deterministic dataflow. The trace construction on event structures is shown to correspond to the least fixed point construction detailed in [6].

Definition 5.1 A span

$$\begin{tikzcd} & E \arrow[dl, "d"'] \arrow[dr, "out"] & \\ A & & B \end{tikzcd}$$

is *deterministic* iff $d(e_1) \uparrow d(e_2) \Rightarrow \neg(e_1 \# e_2)$. □

The extra requirement of the demand morphism ensures that no two conflicting events of E can require consistent configurations of A. Such spans represent stable functions [1] between domains of configurations.

Definition 5.2 Let $A \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B$ be a deterministic span. Define $\widetilde{E}\colon \mathcal{C}(A) \to \mathcal{C}(B)$ by

$$\widetilde{E}\,(y) \overset{def}{=} out\{e \in E \mid d(e) \subseteq y\}\ , \text{ for all } y \in \mathcal{C}(A).$$

□

Theorem 5.3 *The function* $\widetilde{E}$: $\mathcal{C}(A) \to \mathcal{C}(B)$ *is continuous, and stable, for all deterministic spans* E. *Moreover, any stable function from* $\mathcal{C}(A)$ *to* $\mathcal{C}(B)$ *can be represented in this way.*

Proof. As the demand of an event is a finite configuration it follows straightforwardly that $\widetilde{E}$ is continuous. To see it is stable we observe that $\widetilde{E}$ factors into the composition $out \circ F$ where

$$F(y) = \{e \in E \mid d(e) \subseteq y\} \ , \text{ for } y \in \mathcal{C}(A) \ .$$

The function $F : \mathcal{C}(A) \to \mathcal{C}(E)$ preserves all intersections, so certainly stable, while *out* is a rigid morphism so a stable function on configurations, from $\mathcal{C}(E)$ to $\mathcal{C}(B)$. Their composition $\widetilde{E}$ is therefore stable.

To see that any stable function $f : \mathcal{C}(A) \to \mathcal{C}(B)$ can be represented by a deterministic span, define a corresponding event structure E as follows. Its events E are pairs (y, b) where y is a finite configuration of A and b is an event of B for which y a minimal configuration such that $b \in f(y)$—in other words $b \in f(y)$ and for no configuration $y' \subset y$ do we have $b \in f(y')$. (Because f is stable, if $b \in f(w)$ then there is a unique, minimum $y \subseteq w$ with $(y, b) \in E$.) Define

$$(y, b) \leq (y', b') \text{ iff } y \subseteq y' \ \& \ b \leq b' \ , \text{ and}$$
$$(y, b) \# (y', b') \text{ iff } y \not\uparrow y' \ \text{ or } \ b \# b' \ .$$

Define $d(y, b) = y$ and $out(y, b) = b$. It is routine to check that this defines an event structure E, a demand morphism $d : E \to A$ and rigid morphism $out : E \to B$, which together form a deterministic span such that $\widetilde{E} = f$. □

Theorem 5.3 and its proof provide two operations, one from deterministic spans to stable functions, and an inverse operation from stable functions to deterministic spans. These operations are part of an equivalence between the bicategory of deterministic spans and the order-enriched category of stable functions on coherent prime algebraic domains.

Proposition 5.4 *The composition of two deterministic spans is deterministic.*

Proof. Let $A \overset{d_1}{\leftarrow} E_1 \overset{out_1}{\rightarrow} B$ and $B \overset{d_2}{\leftarrow} E_2 \overset{out_2}{\rightarrow} C$ be deterministic spans with composition $A \overset{d}{\leftarrow} E_3 \overset{out}{\rightarrow} C$, which we check is deterministic.

Let (x_1, e_2) and (x_1', e_2') be events in E_3 and assume $d(x_1, e_2) \uparrow \ d(x_1', e_2')$. This is true iff ${d_1}^{\dagger}(x_1) \uparrow {d_1}^{\dagger}(x_1')$. As the span E_1 is deterministic, $x_1 \uparrow x_1'$. Consequently, $d_2(e_2) \uparrow d_2(e_2')$, because by the definition of composition $d_2(e_2) = out_1 \, x_1$ and $d_2(e_2') = out_1 \, x_1'$ and out_1 preserves compatibility. This implies that $\neg(e_2 \# e_2')$, as the span E_2 is deterministic. As $x_1 \uparrow x_1'$ and $\neg(e_2 \# e_2')$ it follows that $\neg((x_1, e_2) \# (x_1', e_2'))$. □

For the remainder of this section, let

$$A \otimes C \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B \otimes C$$

be a deterministic span. We explore the properties of its trace. For notational simplicity we assume that $A \cap C = \emptyset$ and $B \cap C = \emptyset$ and take the parallel compositions $A \otimes C$ and $B \otimes C$ to be unions.

Lemma 5.5 *Let $y \in \mathcal{C}(A)$. For $s \subseteq E$ define*

$$\varphi(s) = \{e \in E \mid d(e) \subseteq y \cup (C \cap out\, s)\} \ .$$

This determines a continuous function $\varphi : \mathcal{C}(E) \to \mathcal{C}(E)$ such that $Sec(y) \overset{def}{=} \mu s.\varphi(s)$, the least fixed point of φ, is a secure configuration of E.

Moreover, $Sec(y)$ is the maximum secure configuration x of E such that

$$d^\dagger(x) \cap A \subseteq y \ .$$

Proof. That φ is a function between configurations follows from determinism, while continuity follows from finiteness of demands, as in the proof of Theorem 5.3.

Then $\mu s.\varphi(s) = \bigcup_{i \in \omega} s_i$, a union of an $\subseteq$-increasing chain of configurations of E given inductively by

$$\begin{aligned} s_0 &= \emptyset \\ s_{i+1} &= \varphi(s_i). \end{aligned}$$

Observe that, as φ is a function from configuration to configuration, $s_i \in \mathcal{C}(E)$ for all $i \in \omega$.

We show by induction that s_i secure for all $i \in \omega$.

Base case: The property trivially holds for s_0.

Inductive step: Assume that s_k is secure. Each event in s_k is secured and therefore has a securing sequence within s_k. Any additional events e in s_{k+1} have $d(e) \cap C \subseteq out\, s_k$. We can therefore give a securing sequence for e by combining the securing sequences of the events upon which it depends according to $\prec_{s_{k+1}}$. Therefore s_{k+1} is secure. This shows that $Sec(y)$ is secure for all $y \in \mathcal{C}$ and it is clear that $d^\dagger(Sec(y)) \cap A \subseteq y$.

Suppose $x \in \mathcal{C}(E)$ is secure and $d^\dagger(x) \cap A \subseteq y$. By Corollary 4.4, $\prec_x$ is well-founded. We show by well-founded induction on $\prec_x$ that

$$\forall e \in x.\ e \in Sec(y) \ .$$

Suppose $e' \in Sec(y)$ for all $e' \prec_x e$. Then $d(e) \cap C \subseteq out\, Sec(y)$ as x is secure and by assumption $d(e) \cap A \subseteq y$; because x is secure $d(e) \cap C \subseteq out\, x$ (Corollary 4.4) and if $out(e') \in d(e) \cap C$ then $e' \to_x e$, so $e' \in Sec(y)$ inductively. Hence $d(e) \subseteq y \cup (C \cap out\, Sec(y))$ and therefore $e \in \varphi(Sec(y)) = Sec(y)$. □

Corollary 5.6 *The trace $Tr(E)$, of the (arbitrary) deterministic span E, is deterministic.*

Proof. Suppose p_1, $p_2 \in Tr(E)$ have compatible demands $d'(p_1)$ and $d'(p_2)$, *i.e.*, there is a configuration y of A such that

$$d^\dagger(p_1) \cap A \subseteq y \text{ and } d^\dagger(p_2) \cap A \subseteq y \ .$$

But $Sec(y)$ is the maximum secure configuration x of E such that

$$d^\dagger(x) \cap A \subseteq y$$

—Lemma 5.5. Both p_1 and p_2 are secure configurations. Hence both $p_1 \subseteq Sec(y)$ and $p_2 \subseteq Sec(y)$, so p_1 and p_2 are compatible as secure configurations, and hence not in conflict as events of the trace. $\square$

Lemma 5.7 *For all $y \in \mathcal{C}(A)$,*

$$\widetilde{Tr(E)}\ (y) = B \cap (out\, Sec(y))\ .$$

Proof. By definition

$$\widetilde{Tr(E)}\ (y) = out'\{p \in Tr(E) \mid d^{\dagger}(p) \cap A \subseteq y\}\ ,$$

where $out'(p) = out(max(p)) \in B$ for $p \in Tr(E)$. If $d^{\dagger}(p) \cap A \subseteq y$ with $p \in Tr(E)$ then p is secure, so $p \subseteq Sec(y)$ by the maximum property of $Sec(y)$—Lemma 5.5. It follows that $\widetilde{Tr(E)}\ (y) \subseteq B \cap (out\, Sec(y))$. To see the reverse inclusion, note that for any $e \in Sec(y)$ with $out(e) \in B$ we have $[e]_{Sec(y)} \in Tr(E)$. $\square$

In conclusion we can understand the trace of the deterministic span in terms of the least fixed point of its associated function; the trace of deterministic event structures reduces to the trace known to Kahn [6].

Theorem 5.8 *For all $w \in \mathcal{C}(A)$,*

$$\widetilde{Tr(E)}\ (w) = B \cap \mu z \in \mathcal{C}(B \otimes C).\ \widetilde{E}\ (w \cup (C \cap out\, z))\ .$$

Proof. By Lemma 5.7,

$$\widetilde{Tr(E)}\ (w) = B \cap out(\mu x.\, \varphi(x))$$

where $\varphi : \mathcal{C}(E) \to \mathcal{C}(E)$ is the continuous function given by

$$\varphi(x) = \{e \in E \mid d(e) \subseteq w \cup (C \cap out\, x)\}$$

for $x \in \mathcal{C}(E)$.

Define the continuous function $\psi : \mathcal{C}(B \otimes C) \to \mathcal{C}(B \otimes C)$ by taking

$$\psi(u) = \widetilde{E}\ (w \cup (C \cap u))$$

for $u \in \mathcal{C}(B \otimes C)$.

It is easy to see that $out : \mathcal{C}(E) \to \mathcal{C}(B \otimes C)$ induces a strict continuous function between the domains of configurations. Directly from their definition we see that

$$out\, \varphi(x) = \psi(out\, x)$$

for $x \in \mathcal{C}(E)$; both expressions equal

$$out\{e \in E \mid d(e) \subseteq w \cup (C \cap out\, x)\}\ .$$

By a well-known property of least fixed points we now know that $\mu u.\,\psi(u) = out(\mu x.\,\varphi(x))$, which gives the result directly. □

6 Concluding Remarks

Spans of event structures provide an intuitive semantics for both deterministic and nondeterministic dataflow. The semantics provides a good example of the way in which event structures can be used to represent both types and processes acting between types.

The spans of event structures used here were first discovered in joint work with Mikkel Nygaard [9]. There they provided an informative, more operational representation of the profunctors which appeared as denotations of affine-HOPLA—an affine language for higher-order processes. (There is a function space construction for spans to accompany the parallel composition $\otimes$.) It was realised later that the same spans also represented the profunctors used in an earlier 'relational' model of nondeterministic dataflow [4,5].

Such spans are part of a much more general picture, the beginnings of which are sketched in [14,12]. But we hope that our treatment of nondeterministic dataflow can stand alone, and in itself make a convincing case for the usefulness of spans of event structures.

References

[1] G. Berry. *Modèles complement adéquats et stables des λ-calculs typés, Thèse de Doctorat d'Etat.* PhD thesis, Universit. Paris VII, 1979.

[2] J. Brock and W. Ackerman. Scenarios: A model of non-determinate computation. *Formalization of programming concepts*, 107, 1981.

[3] Gian Luca Cattani and Glynn Winskel. Profunctors, open maps and bisimulation. As yet unpublished, February 2004.

[4] T. Hildebrandt, P. Panangaden, and G. Winskel. Relational semantics of nondeterministic dataflow, 1998.

[5] T. T. Hildebrandt, P. Panangaden, and G. Winskel. A relational model of nondeterministic dataflow. *MSCS*, 2003.

[6] G. Kahn. The semantics of a simple language for parallel programming. *Information Processing*, 74:471–475, 1974.

[7] S. MacLane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, second edition, 1998.

[8] M. Nielson, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Theoretical Computer Science*, 13(1):85–108, 1981.

[9] M. Nygaard. *Domain Theory for Concurrency.* PhD thesis, University of Aarhus, July 2003.

[10] A. Rabinovich and B. A. Trakhtenbrot. Communication among relations. In M.S. Paterson, editor, *Proceedings of the 6th ICALP*, volume 443 of *LNCS*, pages 294–307. Springer, 1990.

[11] James R. Russell. Full abstraction for nondeterministic dataflow networks. In *FOCS*, pages 170–175, 1989.

[12] G. Winskel. Event structures with symmetry. To appear in the Festschrift for Gordon Plotkin, ENTCS, 2007.

[13] G. Winskel. Event structure semantics for CCS and related languages. In *ICALP '82*, 1982.

[14] G. Winskel. Relations in concurrency (invited talk). *LICS'05*, 2005. Full version available at www.cl.cam.ac.uk/users/gw104/.

A Appendix: spans and rooted profunctors

We show how spans of event structures correspond to certain profunctors and that the trace we have defined on spans coincides with the trace defined on stable profunctors in [4,5]. The profunctors are between partial orders of finite configurations (regarded as categories). We shall write $\mathcal{C}^o(E)$ for the partial order of finite configurations of an event structure E under inclusion.

In describing the profunctor determined by a span it is helpful to have a further characterisation of rigid morphisms between event structures:

Proposition A.1 *Let E_1 and E_2 be event structures. A function f from the set of events E_1 to the set of events E_2 is a rigid morphism $f : E_1 \to E_2$ iff it*

$$\forall x \in \mathcal{C}(E_1).\ fx \in \mathcal{C}(E_2)\ \&\ \forall e, e' \in x.\ f(e) = f(e') \Rightarrow e = e'\ ,\ \textit{and}$$

$$\forall x' \in \mathcal{C}(E_1), y \in \mathcal{C}(E_2).\ y \subseteq fx' \Rightarrow \exists x \in \mathcal{C}(E_1).\ x \subseteq x'\ \&\ fx = y\ .$$

(x is necessarily unique and given by $x = \{e \in x' \mid out(e) \in y\}$.)

Consider a span $A \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B$ between a event structures A and B. Let y be a finite configuration of A and z a finite configuration of B—so y is some particular finite input and z some particular finite output. Define the set

$$\overline{E}(y, z) = \{x \in \mathcal{C}^o(E) \mid d^\dagger(x) \subseteq y\ \&\ out\, x = z\}\ .$$

The set $\overline{E}(y, z)$ consists of all the configurations of E, necessarily finite, which for input y yield output z. The set $\overline{E}(y, z)$ is functorial in $y \in \mathcal{C}^o(A)$ and $z \in \mathcal{C}^o(B)^{\mathrm{op}}$; it respects the inclusion order on configurations covariantly for input and contravariantly for output. Suppose $y \subseteq y'$ in $\mathcal{C}^o(A)$. Then, $\overline{E}(y, z) \subseteq \overline{E}(y', z)$ — simply because any configuration x of E with demand $d^\dagger(x) \subseteq y$ will make $d^\dagger(x) \subseteq y'$. So, an inclusion $i : y \subseteq y'$ in $\mathcal{C}^o(A)$ determines an inclusion map

$$\overline{E}(i, z) : \overline{E}(y, z) \hookrightarrow \overline{E}(y', z)\ .$$

An inclusion $j : z \subseteq z'$ in $\mathcal{C}^o(B)$ determines a map

$$\overline{E}(y, j) : \overline{E}(y, z') \to \overline{E}(y, z)\ .$$

It takes x', for which $out\, x' = z'$, to the unique sub-configuration $x \subseteq x'$ of E for which $out\, x = z$; this exists because out is a rigid map of event structures—Proposition A.1. It is easy to see that these associations of maps with inclusions respect identities and composition, so we have a functor to the category of sets

$$\overline{E} : \mathcal{C}^o(A) \times \mathcal{C}^o(B)^{\mathrm{op}} \to \mathbf{Set}\ ,$$

i.e., a *profunctor* from $\mathcal{C}^o(A)$ to $\mathcal{C}^o(B)$.

By the properties of rigid maps it is easy to see that $\overline{E}(y,\emptyset) = \{\emptyset\}$, so a singleton, for any $y \in \mathcal{C}^o(A)$—the profunctor is *rooted*. Via 'currying' the profunctor corresponds to a functor from $\mathcal{C}^o(A)$ to the functor category $[\mathcal{C}^o(B)^{\mathrm{op}}, \mathbf{Set}]$ of *presheaves* over $\mathcal{C}^o(B)$. It can be shown that the functor so obtained preserves pullbacks. Consequently, the profunctor $\overline{E}$ is *stable* in the sense of [4,5]. It follows that the composition of spans is sent to the composition of (stable) profunctors. In addition, a morphism between two spans $A \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B$ and $A \overset{d'}{\leftarrow} E' \overset{out'}{\rightarrow} B$ is a rigid morphism of event structures $f : E \to E'$ such that the two triangles commute in

$$\begin{tikzcd}
 & E \arrow[d, "f"] \arrow[ddl, "d"'] \arrow[ddr, "out"] & \\
 & E' \arrow[dl, "d'"] \arrow[dr, "out'"'] & \\
A & & B
\end{tikzcd}$$

—recall we regard rigid morphisms and demand morphisms as maps on configurations in making sense of their composition. Such a morphism of spans induce cartesian natural transformations between the associated functors from $\mathcal{C}^o(A)$ to the functor category $[\mathcal{C}^o(B)^{\mathrm{op}}, \mathbf{Set}]$.

The paper [4,5] studies a trace construction on rooted profunctors for nondeterministic dataflow. In *loc. cit.* the tensor operation is also given by simple parallel composition. We now sketch why the trace construction there coincides with that on event structures, that

$$\overline{Tr(E)} \cong Tr(\overline{E})$$

for any span $A \otimes C \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B \otimes C$. In particular, we inherit the properties of the trace proved in the earlier work.

The definition of trace on stable profunctors (Definition 18 in [4]) is based on the usual coend operation on profunctors (see *e.g.,* [3]) but restricted to elements which are secure—where an element being secure is expressed without the benefit of concepts from event structures. We can as well give the definitions with respect to $\overline{E}$ for a particular span of event structures $A \otimes C \overset{d}{\leftarrow} E \overset{out}{\rightarrow} B \otimes C$, assumed from now on. As usual, we will assume that the event structures A and C, and the event structures are disjoint so that the parallel compositions $A \otimes C$ and $B \otimes C$ are given by unions. By definition, an *element* of the profunctor $\overline{E}$ consists of a triple $(y, z; x)$ where $x \in \overline{E}(y, z)$. Elements of $\overline{E}$ can be thought of as states in the computation between A and B. The definition of trace on stable profunctors hinges on three basic relations between elements of the profunctor.

Definition A.2 Let $(y, z; x)$ and $(y', z'; x')$ be elements of $\overline{E}$. Define

$$(y, z; x) \overset{A}{\rightarrow} (y', z'; x') \text{ iff } z = z' \ \&\ x = x' \ \&\ \exists a \in A.\ y \cup \{a\} = y' \ , \text{ and}$$
$$(y, z; x) \overset{B}{\rightarrow} (y', z'; x') \text{ iff } y = y' \ \&\ \exists b \in B.\ z \cup \{b\} = z' \ \&\ x = \{e \in x' \mid out(e) \in z\} \ .$$

Define $(y, z; x) \rightsquigarrow (y', z'; x')$ iff $(y, z'; x')$ is an element with

$$\begin{aligned}&y \subseteq y' \ \&\ z \subseteq z' \ \&\\ &y \cap A = y' \cap A \ \&\ z \cap B = z' \cap B \ \&\\ &y \cap C = z \cap C \ \&\ y' \cap C = z' \cap C \ \&\\ &x = \{e \in x' \mid out(e) \in z\}\ .\end{aligned}$$

□

The relation $\xrightarrow{A}$ concerns the change of element associated with input in A, the relation $\xrightarrow{A}$ with output in B. The relation $\rightsquigarrow$ concerns input in C which is matched by prior output on C and plays a key role in those elements allowed in the definition of trace of a stable profunctor.

We use the following fact several times.

Proposition A.3 *A finite configuration of E is secure iff there is a securing sequence $e_1, \cdots, e_n$ such that $x = \{e_1, \cdots, e_n\}$.*

Proof. *If*: Directly from the definition of securing sequence. *Only if*: Assuming that x is a secure configuration, the relation $\prec_x$ is well-founded and $d^\dagger(x) \cap C \subseteq out\, x$. When x is finite we can order its events, say as $e_1, \cdots, e_n$, in a way that respects $\prec_x$, *i.e.* so $e_i \prec_x e_j$ implies $i < j$. This yields a securing sequence such that $x = \{e_1, \cdots, e_n\}$. □

The notion of *secure element* of a stable profunctor derived from these relations (and used in [4,5]) corresponds to that of secure configuration:

Lemma A.4 *Let $(y, z; x)$ be an element of $\overline{E}$. Then, $(\emptyset, \emptyset; \emptyset)(\xrightarrow{A} \cup \xrightarrow{B} \cup \rightsquigarrow)^\star(y, z; x)$ (i.e., the element is secure [4,5]) iff the configuration x is secure.*

Proof. *Only if*: Consider the following property of an element $(y, z; x)$ of $\overline{E}$:

$$Q(y, z; x) \text{ iff } x \text{ is secure and } y \cap C \subseteq out\, x\ .$$

Clearly $Q(\emptyset, \emptyset; \emptyset)$. It suffices to show in the cases (i) $(y, z; x) \xrightarrow{A} (y', z'; x')$, (ii) $(y, z; x) \xrightarrow{B} (y', z'; x')$ or (iii) $(y, z; x) \rightsquigarrow (y', z'; x')$, that if $Q(y, z; x)$ then $Q(y', z'; x')$.

(i) If $(y, z; x) \xrightarrow{A} (y', z'; x')$, then $x' = x$ and $y' \cap C = y \cap C$. So clearly $Q(y, z; x)$ implies $Q(y', z'; x')$.

(ii) Suppose $(y, z; x) \xrightarrow{B} (y', z'; x')$ and that x is secure and $y \cap C \subseteq out\, x$. Then $y' = y$ and $x' = x \cup \{e'\}$ for some $e' \notin x$ with $out(e') \in B$. Because x is finite and secure we can construct a securing sequence $e_1, \cdots, e_n$ such that $x = \{e_1, \cdots, e_n\}$. The extended sequence $e_1, \cdots, e_n, e'$ is also a securing sequence; the extra requirement that $d(e') \cap C \subseteq out\{e_1, \cdots, e_n\}$ follows by

$$d(e') \cap C \subseteq d^\dagger(x') \cap C \subseteq y' \cap C = y \cap C \subseteq out\, x\ .$$

Hence x' is secure, and clearly $y' \cap C \subseteq out\, x'$.
(*iii*) Suppose $(y, z; x) \rightsquigarrow (y', z'; x')$ and that x is secure and $y \cap C \subseteq out\, x$. As in (*ii*), take a securing sequence $e_1, \cdots, e_n$ such that $x = \{e_1, \cdots, e_n\}$. Because $x \subseteq x'$, with x' a finite configuration, we can choose a sequence of events $e'_1, \cdots, e'_m$ such that $x \cup \{e'_1, \cdots, e'_i\}$ is a configuration of E for all $0 < i \leq m$ with $x' = x \cup \{e'_1, \cdots, e'_m\}$. We claim that the concatenated sequence

$$e_1, \cdots, e_n, e'_1, \cdots, e'_m$$

is a securing sequence: from the definition of $\rightsquigarrow$ it follows that $d^\dagger(x') \subseteq y$, and hence that

$$d(e'_i) \cap C \subseteq y \cap C \subseteq out\, x \subseteq \{e_1, \cdots, e_n, e'_1, \cdots, e'_{i-1}\} \ ,$$

the extra condition required to be securing. Thus x' is secure. From the definition of $\rightsquigarrow$, we have $y' \cap C = z' \cap C$, and $z' \cap C = (out\, x') \cap C$ as $(y', z'; x')$ is an element of $\overline{E}$. So certainly $y' \cap C \subseteq out\, x'$.
If: Write $x \xrightarrow{e} x'$ iff x and x' are secure configurations of E such that $e \notin x$ and $x' = x \cup \{e\}$.

We first show: if $x \xrightarrow{e} x'$ and $(y, z; x)$ is an element of $\overline{E}$ with $z \cap C \subseteq y$, then $(y, z; x)(\xrightarrow{A} \cup \xrightarrow{B} \cup \rightsquigarrow)^\star(y', z'; x')$ for some element $(y', z'; x')$ with $z' \cap C \subseteq y'$.

Assume $x \xrightarrow{e} x'$ and $(y, z; x)$ is an element with $z \cap C \subseteq y$. First note that as x' is secure, $d(e) \cap C \subseteq out\, x$; for a securing sequence $e_1, \cdots, e_n = e$ for e in x' we must have $d(e) \cap C \subseteq \{e_1, \cdots, e_{n-1}\} \subseteq x$. There are two cases, when $out(e) \in B$ and $out(e) \in C$.

Suppose $out(e) \in B$. Then

$$(y, z; x) \xrightarrow{A}{}^* (y \cup d(e), z; x) \xrightarrow{B} (y \cup d(e), z \cup \{out(e)\}; x') \ ,$$

where we have made use of the fact that $d(e) \setminus y \subseteq A$. This follows because

$$d(e) \cap C \subseteq (out\, x) \cap C = z \cap C \subseteq y \ .$$

Also,

$$(z \cup \{out(e)\}) \cap C = z \cap C \subseteq y \subseteq y \cup d(e) \ .$$

Suppose $out(e) \in C$. Then

$$(y, z; x) \xrightarrow{A}{}^* (y \cup d(e), z; x) \rightsquigarrow (y \cup d(e) \cup \{out(e)\}, z \cup \{out(e)\}; x') \ .$$

Again we have used the fact that $d(e) \setminus y \subseteq A$. Also,

$$(z \cup \{out(e)\}) \cap C = (z \cap C) \cup \{out(e)\} \subseteq y \cup d(e) \cup \{out(e)\} \ .$$

Now, assume x is a finite, secure configuration of E. We can choose a securing sequence $e_1, \cdots, e_n$ such that $\{e_1, \cdots, e_n\} = x$. Write $x_i = \{e_1, \cdots, e_i\}$, where $0 \leq i \leq n$. Then each x_i is secure and

$$\emptyset = x_0 \xrightarrow{e_1} x_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} x_n = x \ .$$

By the above we can inductively, left-to-right, produce a chain

$$(\emptyset, \emptyset; \emptyset)(\xrightarrow{A} \cup \xrightarrow{B} \cup \rightsquigarrow)^\star(y, z; x)$$

to an element $(y, z; x)$ of $\overline{E}$. □

Now we define the trace of the stable profunctor $\overline{E}$ from [4,5]. For those familiar with coends, the definition mimics the construction of a coend in the category of sets, but in this case restricted to secured elements. It rests on an equivalence relation $\sim$ between secured elements (the relation coincides with the usual equivalence associated with the coend in sets).

Definition A.5 The relation $\sim$, between secure elements of $\overline{E}$, is defined to be the symmetric, transitive closure of $\rightsquigarrow$.

Following [4,5], the trace of $\overline{E}$ is defined to be the (stable) profunctor

$$Tr(\overline{E}) : \mathcal{C}^o(A) \times \mathcal{C}^o(B)^{\text{op}} \to \textbf{Set} \; ,$$

given as follows. For $y_0 \in \mathcal{C}^o(A)$ and $z_0 \in \mathcal{C}^o(B)$,

$$Tr(\overline{E})(y_0, z_0) = \{\{y, z; x\}_\sim \mid (y, z; x) \text{ is a secure element } \;\&\; y \cap A = y_0 \;\&\; z \cap B = z_0\} \, .$$

An inclusion $i : y_0 \subseteq y_0'$ in $\mathcal{C}^o(A)$ determines a map

$$Tr(\overline{E})(i, z_0) : Tr(\overline{E})(y_0, z_0) \to Tr(\overline{E})(y_0', z_0) \; ,$$

which takes $\{y, z; x\}_\sim$ to $\{y \cup y_0', z; x\}_\sim$. An inclusion $j : z_0 \subseteq z_0'$ in $\mathcal{C}^o(B)$ determines a map

$$Tr(\overline{E})(y_0, j) : Tr(\overline{E})(y_0, z_0') \to Tr(\overline{E})(y_0, z_0) \; ;$$

it takes $\{y', z'; x'\}_\sim$ to $\{y', z' \cap z_0; \{e \in x' \mid out(e) \in z' \cap z_0\}\}_\sim$. (It follows from the functoriality of $\overline{E}$ that the functions $Tr(\overline{E})(i, z_0)$ and $Tr(\overline{E})(y_0, j)$ are well-defined, as does the functoriality of $Tr(\overline{E})$.) □

Definition A.6 Let x be a secure configuration of E. Define

$$min(x) \overset{def}{=} \bigcap \{s \subseteq x \mid s \in \mathcal{C}(E) \;\&\; s \text{ secure} \;\&\; out(s) \cap B = out(x) \cap B\} \; .$$

(Because the secure configuration of E form a stable family, $min(x)$ is the minimum secure sub-configuration of x that outputs the same B events as x.) □

Lemma A.7 *For secure elements,* $(y, z; x) \sim (y', z'; x')$ *iff*

$$y \cap A = y' \cap A \;\&\; z \cap B = z' \cap B \;\&\; min(x) = min(x') \; .$$

Proof. *If*: Assume $y \cap A = y' \cap A$ and $z \cap B = z' \cap B$ and $min(x) = min(x')$. Define $x_0 = min(x)$, $y_0 = d^\dagger(x_0) \cup (y \cap A)$ and $z_0 = out\, x_0$. Then $(y_0, z_0; x_0) \rightsquigarrow (y, z; x)$ from the definition of $\rightsquigarrow$. Similarly, $(y_0, z_0; x_0) \rightsquigarrow (y', z'; x')$. Hence $(y, z; x) \sim (y', z'; x')$. *Only if*: Assume $(y, z; x) \rightsquigarrow (y', z'; x')$. Then, $y \cap A = y' \cap A$ and $z \cap B = z' \cap B$ directly from the definition of $\rightsquigarrow$. Clearly, $x \subseteq x'$. From its definition, $min(x)$ is the least, secure configuration $s \subseteq x$ such that $(out\, s) \cap B = z \cap B$. Hence it is also the least, secure configuration $s \subseteq x'$ such that $(out\, s) \cap B = z' \cap B$, so equal to $min(x')$. □

Theorem A.8 *There is a natural isomorphism*

$$\theta : Tr(\overline{E}) \cong \overline{Tr(E)}$$

with components $\theta_{y_0,z_0} : Tr(\overline{E})(y_0, z_0) \to \overline{Tr(E)}(y_0, z_0)$, *for* $y_0 \in \mathcal{C}^o(A)$ *and* $z_0 \in \mathcal{C}^o(B)$, *given by* $\theta_{y_0,z_0}(\{y, z; x\}_\sim) = \{p \in Tr(E) \mid p \subseteq x\}$.

Proof. Observe that for a secure configuration x of E that

$$min(x) = \bigcup \{p \in Tr(E) \mid p \subseteq x\} \ .$$

In particular, θ_{y_0,z_0} is a well-defined function because $\theta_{y_0,z_0}(\{y, z; x\}_\sim)$ consists of those $p \in Tr(E)$ for which $p \subseteq min(x)$. Tentatively, define an inverse to θ_{y_0,z_0},

$$\varphi_{y_0,z_0} : \overline{Tr(E)}(y_0, z_0) \to Tr(\overline{E})(y_0, z_0)$$

by taking $\varphi_{y_0,z_0}(X) = \{y, z; x\}_\sim$ where $x = \bigcup X$, $y = d^\dagger(x) \cup y_0$ and $z = out\, x$. Then,

$$\begin{aligned}\varphi_{y_0,z_0}\theta_{y_0,z_0}(\{y, z; x\}_\sim) &= \varphi_{y_0,z_0}(\{p \in Tr(E) \mid p \subseteq x\}) \\ &= \{d^\dagger(min(x)) \cup y_0, out\, min(x); min(x)\}_\sim = \{y, z; x\}_\sim \ ,\end{aligned}$$

by the observation, and

$$\theta_{y_0,z_0}\varphi_{y_0,z_0}(X) = \{p \in Tr(E) \mid p \subseteq \bigcup X\} = X \ ,$$

from the properties of primes as X is downwards closed. We omit the fairly routine verification of naturality. □