UNIVERSITY *of York*

This is a repository copy of *A Unary Semigroup Trace Algebra*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/156604/

Version: Accepted Version

---

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

A Unary Semigroup Trace Algebra

Pedro Ribeiro[0000−0003−4319−4872]

Department of Computer Science, University of York, York, YO10 5GH, UK
`pedro.ribeiro@york.ac.uk`

Abstract. The Unifying Theories of Programming (UTP) of Hoare and
He promote the unification of semantics catering for different concerns,
such as, termination, data modelling, concurrency and time. Process cal-
culi like *Circus* and CSP can be given semantics in the UTP using reactive
designs whose traces can be abstractly specified using a monoid trace al-
gebra. The prefix order over traces is defined in terms of the monoid
operator. This order, however, is inadequate to characterise a broader
family of timed process algebras whose traces are preordered instead. To
accommodate these, we propose a unary semigroup trace algebra that
is weaker than the monoid algebra. This structure satisfies some of the
axioms of restriction semigroups and is a right P-Ehresmann semigroup.
Reactive designs specified using it satisfy core laws that have been mech-
anised so far in Isabelle/UTP. More importantly, our results improve the
support for unifying trace models in the UTP.

Keywords: Semantics, process algebra, semigroups, UTP

1 Introduction

The Unifying Theories of Programming (UTP) [1] is a relational framework for
characterising different programming paradigms. It promotes the unification of
semantics, while allowing different aspects, such as data, concurrency, termina-
tion, time, and so on, to be considered individually. Programs are specified via
alphabetised relations in the style of Hehner's Predicative Programming [2].

Behaviour, and concurrency, in the style of CCS, ACP and CSP [3], can be
defined in the UTP via the theory of reactive processes. At its core is the notion
of traces, that is, sequences of events that record the history of interactions.

The time dimension has been considered in different ways [4,5,6,7]. In [4], for
example, traces are sequences of pairs that encode discrete time units. The first
component of a pair records the sequence of events performed during that time,
and the second component the set of events refused at that point. In [7], which
presents a theory that can be used to give semantics in the UTP to the hardware
programming language Handel-C [8] and other synchronous languages, a more
abstract view is provided by a parametric model. It requires that the operators
for addition and subtraction of pairs, and traces, satisfy a set of axioms.

Semantic models employing traces typically define a prefix relation $\leq$ that
specifies how a trace can be augmented, encoding some notion of causality. The

semantics for CSP, for example, is defined using sequences whose prefix relation is a partial order. In that setting, a trace s is a prefix of t, written $s \leq t$, exactly when $s \leq t \Leftrightarrow \exists u \bullet s \frown u = t$, that is, there exists a trace u, such that ($\bullet$) the concatenation ($\frown$) of s with u is t. This led Foster et al. [9] to observe that the prefix order for several trace models can be abstractly defined in terms of a left-cancellative monoid, henceforth referred to as the "monoid trace algebra".

In [7], however, a pair (s, r_0) is a prefix of (t, r_1) exactly when $s \leq t$, but the refusal sets r_0 and r_1 are not constrained. Anti-symmetry is thus not satisfied and so the prefix relation on traces is merely a pre-order. The monoid trace algebra is unsatisfactory for such a theory of synchronous languages. Solving this problem is not only of theoretical interest to establish the commonality between different trace structures, but more importantly enables key results to be reusable in synchronous process algebra, thus promoting unification of models and results, a key goal of the UTP.

Unification in the UTP can be exploited in various ways, namely via subset embeddings, weakest completion semantics [10], Galois connections and parametric theories. The approach pursued in this paper is a contribution to the latter by generalising the theory of reactive processes even further. The main contribution is the definition of a unary left-cancellative semigroup, obtained by introducing a unary function and weakening of the monoid trace algebra axioms. The pairs from [7] are shown to satisfy this structure, as are finite sequences (traces) of such pairs. There is surprisingly little impact on the proofs already established for reactive processes as demonstrated by the mechanisation of our results in Isabelle/UTP [11].

The paper is structured as follows. In Section 2 the theory of reactive processes is introduced, as well as the monoid trace algebra. Our unary semigroup trace algebra is defined in Section 3. Our theory of synchronous algebra is characterised in Section 4. In Section 5 we discuss related work. In Section 6 we summarize the main results and provide pointers for future work.

2 Preliminaries

In the UTP programs are specified by alphabetised relations. Variables are used to define computations, with undashed variables (x) capturing the initial value, and dashed variables (x') capturing the later, or final, value. These can be program variables, or auxiliar variables that capture information such as termination or execution time. A UTP theory is characterised by three components: an alphabet, a set of healthiness conditions, and a set of operators.

For example, in a theory of discrete time we may have variables t and t' of type $\mathbb{N}$ to record time. The relation $t' = t + 1$ describes a computation whereby time is incremented by one time unit. To define the set of valid time-monotonic computations, a function $\mathbf{HC}(P) \mathrel{\widehat{=}} P \wedge t \leq t'$ on predicates can be defined ($\widehat{=}$), so that the set of healthy predicates are the fixed points of $\mathbf{HC}$. When the healthiness conditions are idempotent and monotonic, with respect to the

refinement order $\sqsubseteq$, their image forms a complete lattice, which allows reasoning about recursion.

The theory of reactive processes uses the auxiliary variables ok and ok' to capture stability, $wait$ and $wait'$ to record information about termination, tr and tr' to record the history of interactions with the environment, and ref and ref' to record the possibility of refusing interaction. The variable ok indicates whether the previous process is in a stable state, while ok' records this information for the current process. Similarly, $wait$ records termination for the previous process and $wait'$ for the current process. A process only starts executing in a state where ok and $\neg\,wait$ are $true$. Termination occurs when ok' and $\neg\,wait'$ are $true$.

The interactions with the environment are captured by sequences of events, recorded by tr and tr'. The variable tr records the sequence of events that took place before the current process started, while tr' records all the events that have been observed so far. Finally, ref and ref' record the set of events that may be refused by the process at the start, and currently.

In the theory of synchronous algebra, as already said, tr and tr' are sequences of pairs, where the first component is a sequence of events, and the second is a set of events that may be refused. The variables ref and ref' are not used.

2.1   Monoid trace algebra

To conciliate different trace structures for reactive processes, Foster et al. [9] propose a trace algebra, where tr and tr' are of an abstract type $\mathcal{T}$. Below we reproduce its axioms, where $^\frown$ is concatenation, and ε is the empty trace.

Definition 1 (TA). *A trace algebra $(\mathcal{T}, ^\frown, \varepsilon)$ is a monoid satisfying the axioms:*

$$x \,^\frown (y \,^\frown z) = (x \,^\frown y) \,^\frown z \quad \text{(TA1)} \qquad x \,^\frown y = x \,^\frown z \Rightarrow y = z \quad \text{(TA3)}$$
$$\varepsilon \,^\frown x = x \,^\frown \varepsilon = x \qquad\qquad\quad \text{(TA2)} \qquad\qquad x \,^\frown y = \varepsilon \Rightarrow x = \varepsilon \quad \text{(TA4)}$$

Concatenation is associative (TA1), has the empty trace ε as both a left and right unit (TA2), and is left-cancellative (TA3). Axiom TA4 eliminates "negative traces" by requiring that whenever the concatenation of x and y is the empty trace then x must also be the empty trace. The dual law $x \,^\frown y = \varepsilon \Rightarrow y = \varepsilon$ can be deduced from axioms TA2 and TA4. We observe that while in [9] right-cancellation is also proposed as an axiom, the laws of the algebra, as well as the results established for the theory of reactive processes, as proved so far in Isabelle/UTP[1], do not depend on this axiom, and so we can safely omit it.

Standard finite sequences, for example, $(\text{seq }A, ^\frown, \langle\rangle)$ form a trace algebra, where $^\frown$ is sequence concatenation and $\langle\rangle$ is the empty sequence. Using the two trace algebra operators it is possible to define a trace prefix relation $(x \leq y)$ and a trace subtraction operator $(x - y)$ as reproduced below.

Definition 2 (Trace prefix). $x \leq y \mathrel{\widehat{=}} (\exists z \bullet x \,^\frown z = y)$

Definition 3 (Subtraction). $y - x \mathrel{\widehat{=}} \begin{cases} \iota z \bullet y = x \,^\frown z & if \quad x \leq y \\ \varepsilon & otherwise \end{cases}$

---

[1] https://github.com/isabelle-utp (definitions and lemmas hyper-linked using 🐢).

A trace x is a prefix of y $(x \leq y)$ whenever y can be obtained by concatenating x with some trace z. When $x \leq y$ the subtraction $y - x$ is z whose concatenation with x is y, specified using the definite description operator (ι), as z is unique by TA3, and otherwise $y - x$ is ε so that subtraction is total. In [9] it is shown that $(\mathcal{T}, \leq)$ is a partial order, and that ε is the least element. As mentioned, this is unsuitable for the synchronous algebra, so in Section 3 we pursue a preorder.

2.2   Generalised Reactive Processes

Using the trace algebra, it is possible to define the healthiness conditions that underpin several theories based on reactive processes, reproduced below.

Definition 4 (Generalised Reactive Processes).

$$\mathbf{R1}(P) \triangleq tr \leq tr' \qquad\qquad \mathbf{R2}(P) \triangleq P[\varepsilon, tr' - tr/tr, tr']$$

$$\mathbf{R2c}(P) \triangleq \mathbf{R2}(P) \triangleleft tr \leq tr' \triangleright P \quad \mathbf{R2a}(P) \triangleq \bigsqcap z \bullet P[z, z \,\widehat{\,}\, (tr' - tr)/tr, tr']$$

$$\mathbf{R3}(P) \triangleq I\!I \triangleleft wait \triangleright P \qquad\qquad \mathbf{R}(P) \triangleq \mathbf{R1} \circ \mathbf{R2} \circ \mathbf{R3}(P)$$

R1 requires that a trace can only be extended. **R2** requires processes to be insensitive to the initial trace and is specified by substituting tr in P with the empty trace ε and tr' with the difference $tr' - tr$. Because this difference is only well-defined when $tr \leq tr'$, the version **R2c** proposed by Foster et al. [9] applies **R2** conditionally: $P \triangleleft c \triangleright Q$ is P if c is true, and otherwise is Q. **R2a**, defined using the greatest lower bound $\bigsqcap$, is an alternative for **R2** having the same fixed points. **R3** ensures that P may only start if the previous process has terminated ($\neg\, wait$), and otherwise behaves as the identity $I\!I$, which keeps variables unchanged. This ensures that relational composition is sequential composition. Finally, the theory is characterised by **R**, the composition of all conditions.

While these definitions are applicable to several reactive theories, **R2** and **R2a**, for example, cannot be instantiated for synchronous algebra [7], whose counterparts to **R2** and **R2a** are reproduced below with subscript $_S$. Concatenation ($\widehat{\,}_S$) and subtraction ($-_S$) of their traces are also annotated with $_S$.

Definition 5.

$$\mathbf{R2}_S(P) = P[\langle(\langle\rangle, snd(last(tr)))\rangle, tr' -_S tr/tr, tr']$$

$$\mathbf{R2a}_S(P) = \bigsqcap z \bullet P[z, z \,\widehat{\,}_S\, (tr' -_S tr)/tr, tr'] \wedge snd(last(tr)) = snd(last(z))$$

$\mathbf{R2}_S$ considers the substitution of tr with a sequence whose only element is a pair: the first component is the empty sequence, and the second component is the set of events resulting from taking the second component (snd) of the pair extracted from the *last* element of tr, well-defined when **R1** is applied first.

Clearly the empty trace (ε) of the monoid trace algebra cannot abstractly encode an element that can take several values, such as $snd(last(tr))$. On the other hand, an examination of the algebraic laws satisfied by $\mathbf{R2}_S$, and counterparts to **R1** and **R3** in [7,5], reveals a striking similarity with the laws established for generalised reactive designs, which indicates a similar unification is feasible as we demonstrate in Section 4 using the algebra we define next.

3   Unary semigroup trace algebra

Instead of a fixed empty trace ε, we introduce a total function $\Phi : \mathcal{T} \to \mathcal{T}$ to obtain a unary semigroup $(\mathcal{T}, \frown, \Phi)$. The axioms are defined next in Section 3.1. In Section 3.2 we classify it according to the literature on semigroups. In Section 3.3 we show that the prefix relation is a preorder, and redefine subtraction.

3.1   Axioms

The following axioms can be seen as counterparts to that of the monoid trace algebra, adapted to consider Φ and the fact that the structure is not a monoid.

Definition 6 (USTA). *A unary semigroup trace algebra $(\mathcal{T}, \frown, \Phi)$ is a left-cancellative unary semigroup satisfying the following axioms:*

$$x \frown (y \frown z) = (x \frown y) \frown z \quad \text{(USTA1)} \quad x \frown y = x \frown z \Rightarrow y = z \quad \text{(USTA3)}$$
$$x \frown \Phi(x) = x \quad \text{(USTA2)} \quad x \frown y = \Phi(y) \Rightarrow y = \Phi(y) \quad \text{(USTA4)}$$

Concatenation is associative (USTA1) so that we have a semigroup. Similarly to axiom TA2, we require that $\Phi(x)$ is a right identity with respect to concatenation with x (USTA2). Concatenation is also left-cancellative (USTA3). From these three axioms we can establish that $\Phi(x)$ is a left-unit for concatenation.

Lemma 1. $\Phi(x) \frown y = y$

Proof.

$$x \frown y = x \frown y \qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Axiom USTA2]}$$
$$\equiv (x \frown \Phi(x)) \frown y = x \frown y \qquad\qquad\qquad\quad \text{[Axioms USTA1 and USTA3]}$$
$$\Rightarrow \Phi(x) \frown y = y \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \square$$

Similarly to axiom TA4 of the monoid trace algebra, axiom USTA4 also eliminates "negative traces", but when we draw a parallel between ε and Φ, the shape of USTA4 is different. The requirement on the second operand y of the concatenation $x \frown y$ (rather than the first operand x as in axiom TA4) is sufficiently weak to ensure the prefix relation $\leq$, defined in terms of $\frown$, is not anti-symmetric.

To illustrate that axiom TA4 admits structures whose prefix relation $\leq$ is not anti-symmetric, we consider the following example.

Example 1. Consider $(\mathcal{S}, \gg, id)$, where $\mathcal{S}$ contains at least two distinct elements, $x \gg y \,\hat{=}\, y$ and id is the identity function. Such structure is a unary semigroup trace algebra. We show that $\exists\, a, b : \mathcal{S} \bullet a \leq b \wedge b \leq a \wedge a \neq b$.

Proof.

$$\exists\, a, b : \mathcal{S} \bullet a \leq b \wedge b \leq a \wedge a \neq b \qquad\qquad\qquad\qquad\qquad \text{[Definition of } \leq]$$
$$= \exists\, a, b : \mathcal{S} \bullet (\exists\, z \bullet a \gg z = b) \wedge (\exists\, z \bullet b \gg z = a) \wedge a \neq b \quad \text{[Definition of } \gg]$$
$$= \exists\, a, b : \mathcal{S} \bullet (\exists\, z \bullet z = b) \wedge (\exists\, z \bullet z = a) \wedge a \neq b \qquad\quad \text{[One point rule]}$$
$$= \exists\, a, b : \mathcal{S} \bullet a \neq b \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Assumption]}$$
$$= true \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \square$$

An interesting generalisation of $(\mathcal{S},\gg,id)$ is that $(\mathcal{T},\hat{\ },id)$ satisfies the axioms of a U-semigroup [12, p.102]. In general, Φ is idempotent as we establish next.

Lemma 2. $\Phi(\Phi(x)) = \Phi(x)$

Proof. Using Axiom USTA2.

$$\Phi(x) \hat{\ } \Phi(\Phi(x)) = \Phi(x) \qquad\qquad \text{[Lemma 1]}$$
$$\equiv \Phi(x) \hat{\ } \Phi(\Phi(x)) = \Phi(x) \hat{\ } \Phi(x) \qquad\qquad \text{[Axiom USTA3]}$$
$$\Rightarrow \Phi(\Phi(x)) = \Phi(x) \qquad\qquad \square$$

Moreover, if Φ is constant we can obtain the original monoid trace algebra by having $\forall\, x \bullet \Phi(x) = \varepsilon$.

Theorem 1. *Provided $\forall\, x \bullet \Phi(x) = \varepsilon$, and $(\mathcal{T},\hat{\ },\Phi)$ is a unary semigroup trace algebra, then $(\mathcal{T},\hat{\ },\varepsilon)$ is a monoid trace algebra.*

Proof. Axioms TA1, TA2 and TA3 are trivially satisfied. Axiom TA4 can be satisfied by deduction using USTA2 and USTA4. $\qquad\square$

Thus, the monoid trace algebra can be seen as a specialisation of the algebraic structure we propose. This and other results to follow have been mechanised in Isabelle[2]. Moreover, we have used Isabelle's counter-example generator **nitpick** to ascertain that axioms USTA1-USTA4 are independent. Next we discuss how the new structure can be classified according to the literature on semigroups.

3.2   Semigroup properties

To establish key properties of the algebra, we first propose a lemma that is used in proofs to follow. The application of Φ to a trace obtained by concatenating x and y is equal to $\Phi(y)$ as stated in the lemma below.

Lemma 3. $\Phi(x \hat{\ } y) = \Phi(y)$

Proof. Using Axiom USTA2.

$$(x \hat{\ } y) \hat{\ } \Phi(x \hat{\ } y) = x \hat{\ } y \qquad\qquad \text{[Axioms USTA1 and USTA2]}$$
$$\equiv (x \hat{\ } y) \hat{\ } \Phi(x \hat{\ } y) = (x \hat{\ } y) \hat{\ } \Phi(y) \qquad\qquad \text{[Axiom USTA3]}$$
$$\Rightarrow \Phi(x \hat{\ } y) = \Phi(y) \qquad\qquad \square$$

From Lemmas 1 and 3 we can deduce that Φ distributes over $\hat{\ }$, a property implicitly satisfied by left-cancellative restriction semigroups [13].

The structure is neither a left nor a right-restriction semigroup, as it satisfies only two (LR1 and LR2) out of four axioms [13] of left restriction semigroups, and three (RR1 to RR3) out of four axioms of right-restriction semigroups.

---

[2] https://github.com/isabelle-utp/utp-main/tree/ramics2020s

Theorem 2 (Laws of restriction semigroups).

$$\Phi(x) ^\frown x = x \qquad \text{(LR1)}$$

$$x ^\frown \Phi(x) = x \qquad \text{(RR1)}$$

$$\Phi(\Phi(x) ^\frown y) = \Phi(x) ^\frown \Phi(y) \quad \text{(LR2)}$$

$$\Phi(x ^\frown \Phi(y)) = \Phi(x) ^\frown \Phi(y) \quad \text{(RR2)}$$

$$\Phi(x) ^\frown y = y ^\frown \Phi(x ^\frown y) \quad \text{(RR3)}$$

Proof. (LR1) Using Lemma 1; (RR1) using Axiom USTA2.

(LR2) Using Lemmas 1 and 3; (RR2) using, in addition, Lemma 2.

(RR3)

$$y ^\frown \Phi(x ^\frown y) \qquad\qquad\qquad\qquad \text{[Lemma 3]}$$
$$= y ^\frown \Phi(y) \qquad\qquad\qquad \text{[Axiom USTA2 and Lemma 1]}$$
$$= \Phi(x) ^\frown y \qquad\qquad\qquad\qquad\qquad\qquad \square$$

A fourth axiom of restriction semigroups requires commutativity on the application of Φ with respect to $^\frown$ as $\Phi(x) ^\frown \Phi(y) = \Phi(y) ^\frown \Phi(x)$. It is clear from Lemma 1 that this equality cannot hold. We have, however, that the structure satisfies the axioms of right P-Ehresmann semigroups [14], as established next.

Theorem 3. *$(\mathcal{T}, ^\frown, \Phi)$ is a right P-Ehresmann semigroup.*

$$x ^\frown \Phi(x) = x \qquad \text{(PE1)}$$

$$\Phi(\Phi(x) ^\frown \Phi(y)) = \Phi(y) ^\frown \Phi(x) ^\frown \Phi(y) \quad \text{(PE3)}$$

$$\Phi(x ^\frown y) = \Phi(\Phi(x) ^\frown y) \quad \text{(PE2)}$$

$$\Phi(x) ^\frown \Phi(x) = \Phi(x) \qquad \text{(PE4)}$$

Proof. (PE1) Using Axiom USTA2; (PE2) using Lemmas 1 and 3.

(PE3)

$$\Phi(y) ^\frown \Phi(x) ^\frown \Phi(y) \qquad\qquad\qquad\qquad \text{[Lemma 1]}$$
$$= \Phi(x) ^\frown \Phi(y) \qquad\qquad\qquad\qquad\qquad \text{[Lemma 2]}$$
$$= \Phi(\Phi(x)) ^\frown \Phi(\Phi(y)) \qquad\qquad\qquad \text{[Lemmas 1 and 3]}$$
$$= \Phi(\Phi(x) ^\frown \Phi(y))$$

(PE4) Follows from Lemma 1. $\qquad\qquad\qquad\qquad\qquad\qquad \square$

Despite proposing axioms based on a generalisation of those of the monoid trace algebra, it is pleasing to find that such a construction satisfies the axioms of a known class of semigroups. Next we study the induced prefix relation $\leq$ of the algebra, defined in terms of $^\frown$, and its subtraction operator $-$.

3.3 Prefix and subtraction

The prefix relation can be characterised exactly as in Definition 2. In what follows we study its key algebraic properties, starting by showing that it is a preorder.

Theorem 4. *Provided $(\mathcal{T}, ^\frown, \Phi)$ is a USTA then $(\mathcal{T}, \leq)$ is a preorder. (TP1)*

Proof. (Reflexivity) Using Definition 2 and Axiom USTA2; (Transitivity) Using Definition 2, Axiom USTA1 and predicate calculus.

Moreover, we have that $\leq$ satisfies the following laws, numbered to mirror the laws TP1-TP4 of the monoid trace algebra [9].

Theorem 5 (Trace Prefix Laws).

$$\Phi(x) \leq y \quad \text{(TP2)} \quad x \leq x \frown y \quad \text{(TP3)} \quad x \frown y \leq x \frown z \Leftrightarrow y \leq z \quad \text{(TP4)}$$

Trace $\Phi(x)$ is smaller than any other trace (TP2). Law TP3 states that concatenation constructs larger traces, and Law TP4 states that concatenation is monotonic in its right argument. Next, we introduce the subtraction operator.

Definition 7 (Subtraction). $y - x \cong \begin{cases} \iota z \bullet y = x \frown z & if \quad x \leq y \\ \Phi(x) & otherwise \end{cases}$

Subtraction is defined like in Definition 3 when $x \leq y$, and otherwise is defined as $\Phi(x)$. This deliberate choice of $\Phi(x)$ is essential to ensure that the following laws TS1-TS10 (numbered after the laws TS1-TS8 in [9] as counterparts) hold. Notably absent from the following list is the counterpart to TS2 of the monoid trace algebra, which we discuss in the sequel. It does not hold in this setting, but this bears no impact on the results established for the reactive theory.

Theorem 6 (Trace Subtraction Laws).

$$x - \Phi(y) = x \tag{TS1}$$
$$x - x = \Phi(x) \tag{TS3}$$
$$(x \frown y) - x = y \tag{TS4}$$
$$(x - y) - z = x - (y \frown z) \tag{TS5}$$
$$(x \frown y) - (x \frown z) = y - z \tag{TS6}$$
$$y \leq x \land x - y = \Phi(y) \Leftrightarrow x = y \tag{TS7}$$
$$x \leq y \Rightarrow x \frown (y - x) = y \tag{TS8}$$
$$x \leq y \land x \leq z \Rightarrow (y - x = z - x \Leftrightarrow y = z) \tag{TS9}$$
$$x \leq y \land x \leq z \land z \leq y \Rightarrow (y - x) - (z - x) = y - z \tag{TS10}$$

Law TS1 states that the subtraction of a trace $\Phi(y)$ from another trace is ineffective. Law TS3 states that subtracting a trace from itself is equal to applying Φ. Laws TS4-TS6 and TS8 capture expected properties of concatenation and subtraction, also satisfied by the monoid trace algebra. The implication in Law TS7 states that if the subtraction $x - y$ is $\Phi(y)$, and y is a prefix of x, then x and y are the same. The reverse implication follows from Law TS3. The novel laws TS9-TS10 correspond to axioms SSub:same and SSub:subsub in [7, p.95-96].

The counterpart to Law TS2 ($\varepsilon - x = \varepsilon$) of the monoid trace algebra [9] could be stated in this setting as $\Phi(y) - x = \Phi(y)$. However, this equality does not hold in general. In particular, for example, if $x \leq \Phi(y)$ holds it is not necessarily the case that $(\iota z \bullet \Phi(y) = x \frown z) = \Phi(y)$. Existing proofs for reactive processes do not depend on Law TS2, so the fact that it does not hold in our trace algebra has no practical impact. Next, we focus on instances of the algebra.

4   Trace models

In this section we focus on instances of our trace algebra, and show that it can be instantiated to yield the traces of [7]. To that end, we consider pairs in Section 4.1 whose first component is a USTA. Then in Section 4.2 we consider these pairs as elements of finite non-empty sequences and show the lifted structure is a USTA.

4.1   Parametric pairs

We introduce pairs $\mathcal{P} : \mathcal{H} \times \mathcal{R}$ parametrised by types $\mathcal{H}$ and $\mathcal{R}$, whose $\mathcal{H}$ must be a USTA $(\mathcal{H}, +_{\mathcal{H}}, \Phi_{\mathcal{H}})$ where $+_{\mathcal{H}} : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$ is concatenation, and $\Phi_{\mathcal{H}} : \mathcal{H} \to \mathcal{H}$ is the unary function of the USTA. To construct a USTA for parametric pairs $(\mathcal{P}, +_{\mathcal{P}}, \Phi_{\mathcal{P}})$, we define concatenation of pairs $(+_{\mathcal{P}})$ and $\Phi_{\mathcal{P}}$ as follows.

Definition 8. $(h_1, r_1) +_{\mathcal{P}} (h_2, r_2) = (h_1 +_{\mathcal{H}} h_2, r_2)$

$$\Phi_{\mathcal{P}}(h_1, r_1) = (\Phi_{\mathcal{H}}(h_1), r_1)$$

Concatenation of (h_1, r_1) and (h_2, r_2) is a pair where: the first component is the result of applying $+_{\mathcal{H}}$ to h_1 and h_2, and the second component is r_2. $\Phi_{\mathcal{P}}$ is defined as the application of $\Phi_{\mathcal{H}}$ to the first component. The definition of $+_{\mathcal{P}}$ closely follows the concatenation specified in [7]. However, unlike [7] we do not need to specify subtraction, as instead it can be derived as a lemma below.

Lemma 4. *Provided* $h_2 \leq h_1$, $(h_1, r_1) - (h_2, r_2) = (h_1 - h_2, r_1)$.

With the above construction we can establish that $(\mathcal{P}, +_{\mathcal{P}}, \Phi_{\mathcal{P}})$ is a USTA.

Theorem 7. *Provided* $(\mathcal{H}, +_{\mathcal{H}}, \Phi_{\mathcal{H}})$ *is a USTA then* $(\mathcal{P}, +_{\mathcal{P}}, \Phi_{\mathcal{P}})$ *is a USTA.*

Thus, the pairs of [7] form a USTA. Next, we consider a model for traces constructed from finite non-empty sequences whose elements are pairs of type $\mathcal{P}$.

4.2   Synchronous traces

As already mentioned, the traces of synchronous process algebra consist of non-empty sequences of pairs [7]. In this section we construct this abstract trace structure stepwise, starting by defining a specialised model of finite non-empty sequences that is a USTA. This is then used to lift pairs of type $\mathcal{P}$ to traces.

Traces A trace in this setting is a finite non-empty sequence defined via a recursive data type *fs* below, specified using the Z [15] notation for type constructors.

Definition 9. $fs ::= One \langle\!\langle \sigma \rangle\!\rangle \mid Cons \langle\!\langle \sigma \times fs \rangle\!\rangle$

One constructs a sequence with a single element of type σ, and *Cons* constructs a sequence where an element is followed by a sequence of type *fs*. We use angled brackets $\langle a_0, ..., a_n \rangle_{fs}$ to represent consecutive applications of *Cons*, ending in

One a_n, and $\langle a_0 \rangle_{fs}$ for a single construction *One a_0*. The subscript $_{fs}$ distinguishes finite non-empty sequences from standard finite sequences (that may be empty).

To construct a USTA for an *fs* parametrised by a given type σ that is a USTA $(\sigma, \widehat{}_\sigma, \Phi_\sigma)$, we need to instantiate the respective structure $(fs, \widehat{}_{fs}, \Phi_{fs})$ in terms of $\widehat{}_\sigma$ and Φ_σ. We define concatenation ($\widehat{}_{fs}$) next, and Φ_{fs} in the sequel.

Definition 10 (Concatenation of non-empty sequences).

$$\_\ \widehat{}_{fs}\ \_ : fs \times fs \rightarrow fs$$

$$One\ x\ \widehat{}_{fs}\ One\ y = One\ (x +_\sigma y)$$
$$\forall\, x, y : \sigma;\ f, g : fs \bullet One\ x\ \widehat{}_{fs}\ Cons\,(y, f) = Cons\,(x +_\sigma y, f)$$
$$Cons\,(x, f)\ \widehat{}_{fs}\ g = Cons\,(x, f\ \widehat{}_{fs}\ g)$$

The concatenation of two sequences $\langle x \rangle_{fs}$ and $\langle y \rangle_{fs}$, with one element each, is a sequence whose only element is the result of the sum ($+_\sigma$) of x and y. A sequence $\langle x \rangle_{fs}$ concatenated with $\langle a_0, ..., a_n \rangle_{fs}$ is defined as $\langle x +_\sigma a_0, ..., a_n \rangle_{fs}$, that is, the first element is the sum ($+_\sigma$) of x and the first element a_0 of the second sequence. Finally, a sequence $\langle a_0, ..., a_n \rangle_{fs}$ concatenated with g has a_0 as first element followed by the concatenation of the tail of that sequence with g.

We observe that $\widehat{}_{fs}$ is distinctive from standard sequence concatenation, so as to induce an appropriate definition for prefixing and subtraction (Definitions 2 and 7). For example, the subtraction of $\langle a \rangle_{fs}$ from itself is the sequence z whose concatenation with $\langle a \rangle_{fs}$ yields $\langle a \rangle_{fs}$ ($\iota z \bullet \langle a \rangle_{fs} = \langle a \rangle_{fs}\ \widehat{}_{fs}\ z$). Because *fs* sequences are non-empty, z is the sequence $\langle \Phi_\sigma(a) \rangle_{fs}$ so that $\langle a +_\sigma \Phi_\sigma(a) \rangle_{fs} = \langle a \rangle_{fs}$, as required. This in contrast to subtraction of standard sequences, where $\langle a \rangle - \langle a \rangle = \langle \rangle$. Similar reasoning applies to ensure $\leq$ is reflexive.

Indeed to show that $(fs, \widehat{}_{fs}, \Phi_{fs})$ is a USTA given a type σ that is a USTA $(\sigma, +_\sigma, \Phi_\sigma)$, we define Φ_{fs} in terms of Φ_σ as follows.

Definition 11. $\Phi_{fs}(x) = \langle \Phi_\sigma(last(x)) \rangle_{fs}$

It is defined as the sequence whose only element is obtained by applying Φ_σ to its *last* element. By construction x is non-empty, so *last* and *head* are always well-defined. Thus, provided σ is a USTA, a sequence s of type *fs* can be split into concatenations involving its *front* and *last* element, and its *head* and *tail*.

Lemma 5. $front(s)\ \widehat{}_{fs}\ \langle last(s) \rangle_{fs} = s$, and $\langle head(s) \rangle_{fs}\ \widehat{}_{fs}\ tail(s) = s$.

The functions *front* and *tail* are tailored to non-empty sequences. For example, $front(\langle a \rangle_{fs})$ is $\langle \Phi_\sigma(a) \rangle_{fs}$, while $front(\langle a, b \rangle_{fs})$ is $\langle a, \Phi_\sigma(b) \rangle_{fs}$, and $tail(\langle a \rangle_{fs})$ is $\langle \Phi_\sigma(a) \rangle_{fs}$, while $tail(\langle a, b \rangle_{fs})$ is $\langle \Phi_\sigma(a), b \rangle_{fs}$, so that the decomposition holds.

Next we use this structure to instantiate the USTA for *fs* sequences, which corresponds to the trace structure underlying synchronous process algebra.

Theorem 8. *Provided* $(\sigma, +_\sigma, \Phi_\sigma)$ *is a USTA, then* $(fs, \widehat{}_{fs}, \Phi_{fs})$ *is a USTA.*

As a corollary to this theorem we have that a parametric pair $\mathcal{P}$ whose type parameter $\mathcal{H}$ is a USTA $(\mathcal{H}, +_\mathcal{H}, \Phi_\mathcal{H})$ induces a $(fs, \widehat{}_{fs}, \Phi_{fs})$ USTA.

Corollary 1. *If $(\mathcal{H}, +_{\mathcal{H}}, \Phi_{\mathcal{H}})$ is a USTA, then $(fs, \hat{\,}_{fs}, \Phi_{fs})$ is a USTA.*

This demonstrates that to construct such a USTA it is sufficient to show that $\mathcal{H}$ is a USTA. This is a much more general, and concise, construction, than that proposed in [7], which instead requires satisfying nearly 26 axioms. Moreover, our results do not rely on any assumptions about the type $\mathcal{R}$, thus allowing the second component of such pairs in a trace to record arbitrary information, not only refusal sets as proposed in [7]. Next we focus on key properties of traces leading to a demonstration that we can derive core laws of [7], and the healthiness conditions of the corresponding UTP theory.

Properties Below we establish key results on the difference of *fs* sequences.

Theorem 9. *Provided $(\sigma, +_\sigma, \Phi_\sigma)$ is a USTA and $s \le t$, where $s, t : fs$,*

$$tail(t - s) = tail(t - front(s)) \tag{S1}$$
$$head(t - s) = head(t - front(s)) - last(s) \tag{S2}$$
$$last(s) \le head(t - front(s)) \tag{S3}$$

The *tail* of the difference $t - s$ is the tail of the difference between t and the *front* of s (S1). Likewise, the *head* of the difference $t - s$ is equal to the last element of s subtracted from the *head* of the difference $t - front(s)$ (S2). Related, (S3) establishes that $last(s)$ is a prefix of $head(t - front(s))$.

To illustrate the role of $S1$, we consider, as an example the subtraction of a *fs* sequence whose elements are standard sequences. The subtraction of $\langle \langle a \rangle, \langle b \rangle \rangle_{fs}$ from $\langle \langle a \rangle, \langle b, c \rangle, \langle d \rangle \rangle_{fs}$ is $\langle \langle c \rangle, \langle d \rangle \rangle_{fs}$, indicating that the first element where the sequences differ is the inner sequence $\langle b, c \rangle$. The *front* of $\langle \langle a \rangle, \langle b \rangle \rangle_{fs}$ is $\langle \langle a \rangle, \langle \rangle \rangle_{fs}$, and so the difference $\langle \langle a \rangle, \langle b, c \rangle, \langle d \rangle \rangle_{fs} - \langle \langle a \rangle, \langle \rangle \rangle_{fs}$ is $\langle \langle b, c \rangle, \langle d \rangle \rangle_{fs}$. Finally, the $tail(\langle \langle b, c \rangle, \langle d \rangle \rangle_{fs}) = \langle \langle \rangle, \langle d \rangle \rangle_{fs}$ coincides with that of $\langle \langle c \rangle, \langle d \rangle \rangle_{fs}$.

Moreover, we show below that Eq. 3 in [7] also holds in our setting of *fs* sequences of parametric pairs $\mathcal{P}$, provided $(\mathcal{H}, +_{\mathcal{H}}, \Phi_{\mathcal{H}})$ is a USTA.

Lemma 6. $s \,\hat{\,}_{fs}\, t = front(s) \,\hat{\,}_{fs}\, \langle last(s) +_{\mathcal{P}} head(t) \rangle_{fs} \,\hat{\,}_{fs}\, tail(t)$.

The concatenation of traces s and t can be decomposed into the concatenation of the *front* of s with a singleton sequence, whose only element is the result of concatenating $(+_{\mathcal{P}})$ the *last* pair of s and the *head* pair of t, and the *tail* of t.

Reactive processes Besides the definition of an abstract trace structure that can be instantiated to yield the trace structure in [7], we discuss next how it can be used to define a generalised theory of reactive processes. Here we focus on the instantiation of the healthiness conditions.

Healthiness conditions The functions **R1** and **R3** are stated like in Section 2, but in the context of a USTA $(\mathcal{T}, \hat{\,}, \Phi)$, with tr and tr' of type $\mathcal{T}$. **R2**, on the other hand, must be adapted to accommodate the function Φ.

Definition 12. $\mathbf{R2}(P) = P[\varPhi(tr), tr' - tr/tr, tr']$

$$\mathbf{R2a}(P) = \bigsqcap z \bullet P[z, z \frown (tr' - tr)/tr, tr'] \wedge \varPhi(tr) = \varPhi(z)$$

Our definition for **R2** is stated by replacing ε with $\varPhi(tr)$. Moreover, the definition for **R2a**, when compared to Definition 4, requires that, in addition z and tr agree on the application of $\varPhi$. This closely follows a solution proposed in [7, p. 83].

Despite employing a weaker trace algebra, the core properties of **R1**, **R2** and **R3**, namely idempotency and monotonicity with respect to refinement, continue to hold. Similarly, all laws of reactive processes, and those for other theories built upon reactive processes, namely CSP, continue to hold as demonstrated by the mechanisation in Isabelle/UTP, which features several hundreds of theorems.

Because the existing theories are mechanised we have been able to quickly establish that all relevant properties hold when using our algebra. Proofs of closure for sequential and parallel composition under **R2** required small adjustments to take into account $\varPhi$, but were structurally kept unchanged. Next, we illustrate a concrete instantiation of the algebra to accommodate the trace model of [4].

Concrete instantiation for Circus Time In what follows we show how our algebra can be instantiated to yield the theory of *Circus Time*, that encompasses behaviour and data modelling in a discrete-time setting.

The parametric pair type $\mathcal{P}$ is instantiated with $\mathcal{H}$ as seq $\varSigma$, where $\varSigma$ is a given type of events, which is a USTA (seq $\varSigma, \frown, \langle\rangle$). Concatenation ($\frown$) is associative (USTA1), left-cancellative (USTA3) and satisfies USTA4. The empty sequence $\langle\rangle$ is a right-unit (USTA2). The parameter $\mathcal{R}$ is instantiated as $\mathbb{P}\,\varSigma$, a set of events. Thus, the first component of such a pair is a sequence and the second a set of events. For example, the pair $(\langle a, b\rangle, \{a\})$ records that having performed events a, and then b, the system can refuse to engage in event a.

Therefore, the lifted structure of finite non-empty sequences fs parametrised by the concrete pair structure above, gives rise to a USTA (Corollary 1). For example, in *Circus Time* the sequence $\langle(\langle a, b\rangle, \{a\}), (\langle\rangle, \varSigma)\rangle_{fs}$ encodes a situation where: during the first time unit a and b are performed, with a then being refused, and during the following time unit no events are performed ($\langle\rangle$) with the system refusing to engage in any event ($\varSigma$).

Compared with the approach in [4], we have that both concatenation and subtraction of fs sequences (using the lifted structure) is total and closed under the correct type. This provides for a precise encoding of the healthiness conditions proposed in [4] using our abstract algebra. Furthermore, this makes mechanisation of the model in Isabelle/UTP an easier endeavour by eliminating the need to reprove a substantial base of existing theorems of reactive processes.

In the remainder of this section we show two key results that demonstrate **R1** and **R2** can be instantiated to yield the counterpart definitions for *Circus Time*.

Lemma 7. $s \leq t \Rightarrow front(s) \leq t \wedge fst(last(s)) \leq fst(head(t - front(s)))$.

This corresponds to the conjunct in the definition of **R1** for *Circus Time* as defined in [16], for example, with the understanding that here *front* is total, whereas in [16,7] it is a partial function over standard sequences.

The definition of $\mathbf{R2}_S$ for *Circus Time* is derived next. First we establish a result for the subtraction of *fs* traces that depends on the following lemma.

Lemma 8. $s \leq t \Rightarrow snd(head(t-front(s))-last(s)) = snd(head(t-front(s)))$.

This lemma states that the second component of the difference, between the *head* of the difference t and the *front* of s, and $last(s)$, does not depend on $last(s)$, a result that follows from Lemma 4. For example, the subtraction of $\langle(\langle a\rangle, r_2)\rangle_{fs}$ from $\langle(\langle a, b\rangle, r_1)\rangle_{fs}$ yields the sequence $\langle(\langle b\rangle, r_1)\rangle_{fs}$ as the second component only depends on r_1, but not r_2. Next, we establish a general result for subtraction of *fs* traces.

Theorem 10. *Provided $s \leq t$, where $s, t : fs$,*

$$t - s = \left\langle \begin{pmatrix} fst(head(t-front(s))) - fst(last(s)), \\ snd(head(t-front(s))) \end{pmatrix} \right\rangle_{fs} \mathbin{\widehat{}}_{fs} tail(t-front(s))$$

Proof.

$t - s$ [Lemma 5]

$= \langle head(t-s)\rangle_{fs} \mathbin{\widehat{}}_{fs} tail(t-s)$ [S2 in Theorem 9]

$= \langle head(t-front(s)) - last(s)\rangle_{fs} \mathbin{\widehat{}}_{fs} tail(t-s)$ [S1 in Theorem 9]

$= \langle head(t-front(s)) - last(s)\rangle_{fs} \mathbin{\widehat{}}_{fs} tail(t-front(s))$ [Pair structure]

$$= \left(\left\langle \begin{pmatrix} fst(head(t-front(s)) - last(s), \\ snd(head(t-front(s)) - last(s) \end{pmatrix} \right\rangle_{fs} \atop \mathbin{\widehat{}}_{fs} tail(t-front(s)) \right) \quad \text{[Lemma 4]}$$

$$= \left(\left\langle \begin{pmatrix} fst(head(t-front(s))) - fst(last(s)), \\ snd(head(t-front(s))) - last(s) \end{pmatrix} \right\rangle_{fs} \atop \mathbin{\widehat{}}_{fs} tail(t-front(s)) \right) \quad \text{[Lemma 8]}$$

$$= \left(\left\langle \begin{pmatrix} fst(head(t-front(s))) - fst(last(s)), \\ snd(head(t-front(s))) \end{pmatrix} \right\rangle_{fs} \atop \mathbin{\widehat{}}_{fs} tail(t-front(s)) \right) \quad \square$$

The subtraction $t-s$ can be expressed in terms of the difference $t-front(s)$, and $last(s)$. The *head* of $t-front(s)$ contains the observations up until the end of the current time unit [16, p.13]. Together with the pair instantiation as before we can derive the concrete definition of $\mathbf{R2}$ for *Circus Time*, similarly to Definition 5 where $\Phi(tr)$ becomes $\langle(\langle\rangle, snd(last(tr)))\rangle_{fs}$ following Definitions 8 and 11, and $tr' -_S tr$ is as given by Theorem 10.

5   Related work

Traces are at the core of semantic models for reasoning about causality. Already in Hoare's CSP book [17] we can find a rich collection of operators and laws for manipulating traces. In the standard semantics [3] of CSP traces are sequences

of events ordered by sequence prefixing. Richer semantic models for CSP, such as refusal testing [18,19] and the finite-linear models [3, p.256], also record in traces the set of events refused, or accepted, in the latter, before each event.

The modelling of time in semantics for process algebra is often achieved by associating events or state observations with time. Hayes' reactive timed designs [20], comparable to action systems and TLA, define traces as mappings from time (discrete or continuous) to the values of program variables.

Sherif et al. [4] defined a semantics for *Circus Time* where traces are sequences whose elements are pairs, recording the events performed, and subsequently refused, during a time unit. Wei et al. [5] considered an equivalent model, where events and refusals are recorded separately in two distinct traces of equal length. Woodcock et al. [6] in their semantics for CML define sequences whose elements are events or refusal sets, that implicitly mark the passage of time, a structure pioneered by Lowe and Ouaknine [21] in their timed traces.

Butterfield et al. [7] proposed a parametric theory, which is the inspiration for the work presented in this paper. It generalises the model of *Circus Time* [4] to account for different observation models within a time unit. A similar approach is pursued by Zhu et al. [22], in their semantics for SystemC, who define a trace as a three dimensional sequence structure to account for macro and micro time.

Trace models for true concurrency in process algebra include the works of Barnes [23] and Smith [24]. The latter [24] defines traces whose elements are sequences, with the prefix relation allowing permutations of the inner sequences. This model can likely be instantiated as a trace algebra with elements as sets. Barnes' SCSP, on the other hand, cannot be instantiated within the setting of [7].

More recently, Foster et al. [9] proposed a left-cancellative monoid trace algebra which is at the core of the mechanisation of several reactive theories in Isabelle/UTP [11]. This enabled Foster et al. [25] to define reactive contracts, as well as a theory for hybrid relations [26]. Their prefix relation over traces, however, is an order, which is inadequate to characterise traces where the relation is not anti-symmetric. Our results are complementary and support the unification of further trace models under the Isabelle/UTP framework.

6   Conclusions

Originally motivated by the goal of mechanising *Circus Time* [4] in the theorem prover Isabelle/UTP [11], we have pursued an ambitious generalisation of the monoid trace algebra [9] to account for a broader family of timed process algebras. We have weakened the monoid axioms, inspired by the observations in [7], to construct a novel unary semigroup trace algebra that is also a right P-Ehresmann semigroup. Compared to the large set of axioms in [7], we have a much smaller set that closely mirrors the axioms of the monoid trace algebra.

Our results support the definition of a parametric UTP theory of reactive processes that abstractly characterises several trace-based semantics. Besides the trace models discussed in [9] our algebra can be instantiated to account for the models discussed in [7,8], including *Circus Time* [4]. In the future, we hope

to accommodate the semantics for the system-level language SystemC [22], and perhaps even other synchronous languages such as Esterel [27].

Besides providing a foundation for the unification of further trace models in the UTP, we have also shown that our work has practical impact via its mechanisation in Isabelle/UTP [11]. It promotes the reuse of a large collection of theorems already established for the theories of reactive processes and reactive designs. It would be interesting, for example, to revisit our mechanisation of a stepwise construction for *Circus Time* [16] in this setting. Another avenue for future work is the mechanisation of the Galois connection in [4] that enables timed models to be verified using untimed tools.

The mechanisation of the timed operators of timed process calculi is likely to benefit from the definition of a timed trace algebra, consisting of an additional function from traces to time, with continuous and discrete versions. Basic processes, such as event prefixing and delay, may also be defined parametrically.

We envision it may be feasible to weaken the unary semigroup trace algebra even further to characterise additional trace structures, such as those of refusal-testing, the finite-linear model, and those of SCSP. However, it is likely that such weakenings may reveal certain laws of reactive processes no longer hold. An open question is the treatment of infinite traces, for example, which seem necessary to give a full account of the hiding operator of CSP. The Isabelle/UTP [11] mechanisation will facilitate the design space exploration of such weakenings, with immediate feedback provided to the proof engineer, a facility we used extensively during the course of developing the algebra presented in this paper.

References

1. Hoare, C.A.R., Jifeng, H.: Unifying Theories of Programming. Prentice-Hall (1998)
2. Hehner, E.C.R.: Predicative programming part I. Commun. ACM **27**(2) (February 1984) 134–143
3. Roscoe, A.W.: Understanding concurrent systems. Springer (2010)
4. Sherif, A., Cavalcanti, A.L.C., He, J., Sampaio, A.C.A.: A process algebraic framework for specification and validation of real-time systems. Formal Aspects of Computing **22**(2) (2010) 153–191
5. Wei, K., Woodcock, J.C.P., Cavalcanti, A.L.C.: *Circus Time* with Reactive Designs. In Wolff, B., Gaudel, M.C., Feliachi, A., eds.: Unifying Theories of Programming. Volume 7681 of Lecture Notes in Computer Science., Springer (2013)
6. Woodcock, J., Bryans, J., Canham, S., Foster, S.: The COMPASS modelling language: Timed semantics in UTP. Communicating Process Architectures. Open Channel Publishing (2014)
7. Butterfield, A., Sherif, A., Woodcock, J.C.P.: Slotted *Circus*: A UTP-family of reactive theories. In: Integrated Formal Methods. Volume 4591 of LNCS., Springer-Verlag (2007) 75–97

8. Butterfield, A.: A denotational semantics for Handel-C. Formal Aspects of Computing **23**(2) (Mar 2011) 153–170
9. Foster, S., Cavalcanti, A., Woodcock, J., Zeyda, F.: Unifying theories of time with generalised reactive processes. Information Processing Letters **135** (2018) 47 – 52
10. Woodcock, J., Cavalcanti, A., Foster, S., Mota, A., Ye, K.: Probabilistic semantics for robochart. In Ribeiro, P., Sampaio, A., eds.: Unifying Theories of Programming, Cham, Springer International Publishing (2019) 80–105
11. Foster, S., Zeyda, F., Woodcock, J.: Isabelle/UTP: A mechanised theory engineering framework. In Naumann, D., ed.: Unifying Theories of Programming, Cham, Springer International Publishing (2015) 21–41
12. Howie, J.M.: Fundamentals of semigroup theory. Volume 12. Clarendon Oxford (1995)
13. Cornock, C., Gould, V.: Proper two-sided restriction semigroups and partial actions. Journal of Pure and Applied Algebra **216**(4) (2012) 935 – 949
14. Jones, P.R.: A common framework for restriction semigroups and regular *-semigroups. Journal of Pure and Applied Algebra **216**(3) (2012) 618 – 632
15. Woodcock, J.C.P., Davies, J.: Using Z – Specification, Refinement, and Proof. Prentice-Hall (1996)
16. Ribeiro, P., Cavalcanti, A., Woodcock, J.: A stepwise approach to linking theories. In Bowen, J.P., Zhu, H., eds.: Unifying Theories of Programming, Cham, Springer International Publishing (2017) 134–154
17. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)
18. Mukarram, A.: A refusal testing model for CSP. PhD thesis, University of Oxford (1993)
19. Phillips, I.: Refusal testing. Theoretical Computer Science **50**(3) (1987) 241–284
20. Hayes, I.J., Dunne, S.E., Meinicke, L.: Unifying theories of programming that distinguish nontermination and abort. In Bolduc, C., Desharnais, J., Ktari, B., eds.: Mathematics of Program Construction, Berlin, Heidelberg, Springer Berlin Heidelberg (2010) 178–194
21. Lowe, G., Ouaknine, J.: On timed models and full abstraction. Electronic Notes in Theoretical Computer Science **155** (2006) 497–519
22. Zhu, H., He, J., Qin, S., Brooke, P.J.: Denotational semantics and its algebraic derivation for an event-driven system-level language. Formal Aspects of Computing **27**(1) (Jan 2015) 133–166
23. Barnes, J.E.: A mathematical theory of synchronous communication. University of Oxford (1993)
24. Smith, M.L.: A unifying theory of true concurrency based on CSP and lazy observation. In: Communicating Process Architectures 2005: WoTUG-28: Proceedings of the 28th WoTUG Technical Meeting, 18-21 September 2005, Technische Universiteit Eindhoven, the Netherlands. Volume 63., IOS Press (2005) 177
25. Foster, S., Cavalcanti, A., Canham, S., Woodcock, J., Zeyda, F.: Unifying theories of reactive design contracts. Theoretical Computer Science (2019)
26. Foster, S.: Hybrid relations in Isabelle/UTP. In Ribeiro, P., Sampaio, A., eds.: Unifying Theories of Programming, Cham, Springer International Publishing (2019) 130–153
27. Berry, G., Gonthier, G.: The Esterel synchronous programming language: design, semantics, implementation. Science of Computer Programming **19**(2) (1992) 87 – 152