An interval logic for stream-processing functions: A convolution-based construction

Brijesh Dongol

Department of Computer Science, Brunel University, UK

Abstract. We develop an interval-based logic for reasoning about systems consisting of components specified using stream-processing functions, which map streams of inputs to streams of outputs. The construction is algebraic and builds on a theory of convolution from formal power series. Using these algebraic foundations, we uniformly (and systematically) define operators for time- and space-based (de)composition. We also show that Banach's fixed point theory can be incorporated into the framework, building on an existing theory of partially ordered monoids, which enables a feedback operator to be defined algebraically.

1 Introduction

Many systems (e.g., hybrid systems) require logics that are capable of reasoning about both discrete and continuous behaviours; scalability in reasoning methods for such systems has long been an open challenge. Especially difficult is a logic that enables reasoning about time- and space-based properties, including feedback, to be (de-)composed in a uniform manner. From a uniformity perspective, one way forward is the development of logics and reasoning frameworks from algebraic foundations [12].

In this paper, we build on our previous work on *convolution* [8], which is a concept taken from *formal power series* [9, 2]. Essentially, convolution defines multiplication for functions of type $Q^M = M \to Q$, where M is a partial monoid (see Section 3) and Q is a quantale (see Section 5). For any $x \in M$, the convolution of $f, g \in Q^M$ is given by

$$(f \cdot g)\, x \;=\; \sum_{x=y\circ z} f\, y \odot g\, z.$$

That is, multiplication $\cdot$ at the level of the functions f and g is defined as the sum of all possible decompositions of the argument x into components y and z, where $x = y \circ z$ and each term in the sum is obtained by applying f to y and g to z, then multiplying the results of the function applications using $\odot$.

There are many possible instantiations of M and Q, which allows the algebra to capture many different models of computation (see [8] for details). As we shall see, in this paper, the quantale Q that we consider is a boolean quantale, and M itself has a richer algebraic structure. In particular, we use a monoidal structure M consisting of three different multiplication operators: one for (de)composing
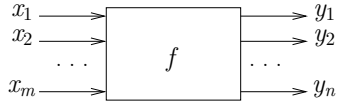
Fig. 1. (m, n)-ary stream-processing function

time, and two for different types of functional (de)composition. We show that by lifting each of these multiplications using convolution results in a tri-quantale over Q^M.

From these algebraic foundations, we construct a new logic for a computation model, suited for reasoning about stream-based systems (e.g., hybrid systems). The logic combines *interval-based reasoning* [19, 16, 14] with *stream-processing functions* [17, 3], where components are modelled by functions from streams of inputs to streams of outputs (see Fig. 1). A basic form of this logic has already been described [8, 14], but this existing treatment does not distinguish between inputs and outputs. As such, the basic form is unable to cope with functional composition and feedback. The extended logic in this paper copes with both in a straightforward manner, while retaining the generality of the previous approach [8]. We discuss possible variations of our logic throughout this paper.

This paper is structured as follows. Section 2 introduces our target computation model of stream-processing functions and Section 3 discusses the algebraic structure, which is used to define pipelined and parallel composition. Section 4 presents a method for reasoning about feedback, adapting Cataldo et al.'s algebraic constructions [4]. Section 5 provides further algebraic background (quantales and convolution), which we use in Section 6 to develop our full logic, consisting of both intervals and stream-processing functions. Section 7 describes method for reasoning about modalities and Section 8 concludes and discusses future work.

2 Stream-processing functions

We aim to reason about systems that evolve over time. These may be modelled by *streams*, which are total functions of type $T \rightarrow X$, where X denotes the (potentially infinite) set of values and $(T, \leq)$ is a linearly ordered set, denoting times. It is well known that T can be instantiated to, for instance, $\mathbb{Z}$ to reason about discrete systems and $\mathbb{R}$ to reason about hybrid systems [5, 4, 10].

Systems may take more than one input stream and produce multiple output streams. If $X_i \subseteq X$ is a set of values, we let $X^{T,m}$ denote $X_1^T \times X_2^T \times \cdots \times X_m^T$. Thus, each $x \in X^{T,m}$ is an m-tuple and each x_i is a stream over type X_i. An (m, n)-ary *stream-processing function* with m input and n output streams is a function $f : X^{T,m} \rightarrow Y^{T,n}$. Note that streams (and hence stream-processing functions) do not contain variables; stream-processing functions simply take an m-tuple of input values and transform them into an n-tuple of output values.

Although a stream-processing function (of type X^T) defines values over all time in T, reasoning typically only takes place after initialisation. For convenience, we assume $0 \in T$ and that stream-processing functions are initialised at time 0.

One of the benefits of using stream-processing functions (which naturally distinguish between input/output streams) is that they simplify reasoning about feedback. In order to ensure feedback is well defined, we require that the streams are κ-causal, with some delay κ. A stream-processing function is *causal* iff its input until time $t \geq 0$ completely determines its output until time t, and is κ-*causal* iff its input until time $t \geq 0$ completely determines its output until time $t + \kappa$ (where $\kappa > 0$). (Delayed) causality imposes the basic requirement that a system cannot anticipate the future values of its inputs. These concepts are formalised below. We use notation $f =_t g$ to denote $\forall u \in T . u \leq t \Rightarrow f\,u = g\,u$, where, following algebraic conventions, we write $f\,x$ for function application $f(x)$.

Definition 1. *Let f be an (m, n)-ary stream-processing function. We say f is* causal *iff*

$$\forall x, x' \in X^{T,m}, t \in T_{\geq 0} \;.\; (x =_t x') \Rightarrow (f\,x =_t f\,x')$$

and that f is κ-causal *with delay* $\kappa > 0$ *iff*

$$\forall x, x' \in X^{T,m}, t \in T_{\geq 0} \;.\; (x =_t x') \Rightarrow (f\,x =_{t+\kappa} f\,x').$$

We will refer to a causal stream-processing function as a *behaviour* and a κ-causal stream-processing function as a *delayed behaviour*.

Example 2. Suppose the temperature of a fridge is given by a stream *temp* (whose behaviour is unspecified for now). A controller that turns the motor on/off to keep the temperature between K_{max} and K_{min} can be modelled by a delayed behaviour:

$$C(temp) = \lambda t : T \;.\; \begin{cases} on & \text{if } temp\,(t - \kappa) > K_{max} \wedge t \geq \kappa \\ off & \text{if } temp\,(t - \kappa) < K_{min} \vee 0 \leq t < \kappa \\ C\,temp\,(t - \kappa) & \text{otherwise} \end{cases}$$

The disjunct $0 \leq t < \kappa$ in the second case defines the initial value of the motor (upto time κ). □

A possible behaviour of the system from Example 2 is given below.
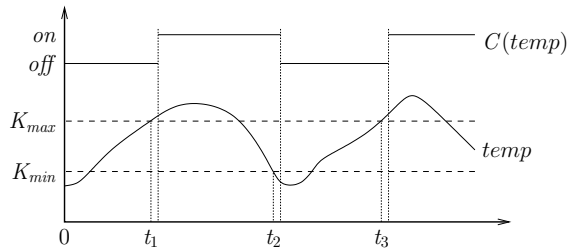
The temperature $temp$ fluctuates between K_{max} an K_{min}. The stream processing function C takes $temp$ as input and transforms it into some output $C(temp)$ resulting in the values on or off. Note the delay κ between the value of $temp$ rising above K_{max} (e.g., at t_1) and the output on, as well as the value of $temp$ dipping below K_{min} (e.g., at t_2) and the output off.

3   Composition algebraically

It is straightforward to see that various composition operators can be defined for stream-processing functions [3, 17], e.g., pipelined composition (see Fig. 2) as well as parallel composition (see Fig. 3). This section describes an algebraic construction, where compositions are defined at the level of partial monoids, and later instantiated to obtain compositions for our computation model of stream-processing functions. In Section 6, we show how our algebraic theory (based on convolution), can be used to lift these structure to the level of specifications. First, we recap our algebraic theory.

Partial Monoids and Bi-Monoids. A *partial monoid* is a structure $(M, \circ, D, E)$ such that M is a set (known as the *carrier set* of the algebra), $D \subseteq M \times M$ the domain of composition, and $\circ : D \to M$ a partial operation of composition. Composition is associative, $x \circ (y \circ z) = (x \circ y) \circ z$, in the sense that if either side of the equation is defined then so is the other and both sides are equal. Furthermore, $E \subseteq M$ is a set of (generalised) units, where for each $x \in M$ there exist $e, e' \in E$ such that $e \circ x = x = x \circ e'$. We follow the convention of leaving out the D from the signature of the partial monoids under consideration, where possible.

Example 3 (Ordered Pairs). Consider the Cartesian product $A \times A$ over a set A. Define

$$D_{OP} = \{(p, q) \in (A \times A) \times (A \times A) \mid \pi_2\, p = \pi_1\, q\}$$

where π_i is the projection onto the ith component of the given tuple. Let $E_{OP} = \{(a, a) \mid a \in A\}$. Define the *cartesian fusion product* $p \gg q = (\pi_1\, p, \pi_2\, q)$. In the presence of D_{OP}, the operator $\gg$ composes two ordered pairs whenever the second coordinate of the first one is equal to the first coordinate of the second one. This turns $(A \times A, \gg, D_{OP}, E_{OP})$ into a partial monoid.   □

The definitions of monoids generalise to n operations. For example, for $n = 2$, a partial *bi-monoid* is a structure $(M, \circ_1, \circ_2, E_1, E_2)$ such that $(M, \circ_1, E_1)$ and $(M, \circ_2, E_2)$ are partial monoids.

Pipeline and parallel composition. To use this algebraic theory, it is simpler to view each stream-processing function as sets of input/output pairs, where a function $f : X \to Y$ is represented by a set of pairs $\{(x, y) : X \times Y \mid x \in dom\, f \wedge y = f\, x\}$. The carrier set F for our algebra is defined as follows. Let
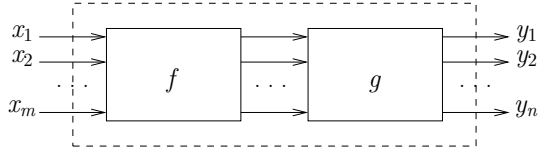
Fig. 2. Pipelined composition $f \gg g$

$F_{m,n} = X^{T,m} \times Y^{T,n}$ be the set of all (m, n)-ary input/output tuples and let $F = \bigcup_{m,n:\mathbb{N}} F_{m,n}$ be the set of all input/output tuples. Also let id be the identity function.

Pipeline composition takes all output messages from the first component and uses them as inputs to the second (see Fig. 2).

Lemma 4 (Pipeline composition). $(F, \gg, id)$ *is a partial monoid with definedness relation* D_{OP}.
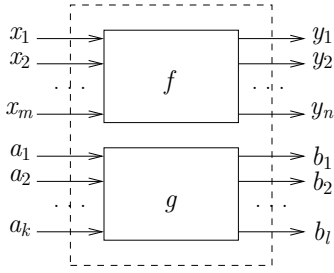

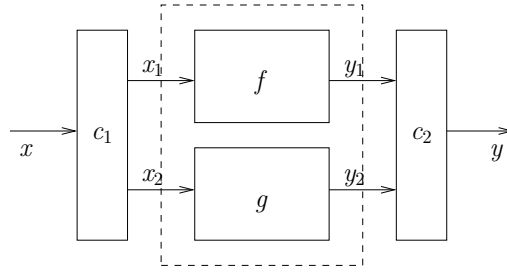**Fig. 3.** Parallel composition $f \otimes g$ **Fig. 4.** Duplicating/combining inputs/outputs

Parallel composition (see Fig. 3) of stream-processing functions simply constructs a new tuple, combining the first and second arguments to the multiplication. The proof of this lemma is straightforward. We use notation $x \frown y$ to denote concatenation for tuples x and y and $\langle \rangle$ to denote the empty tuple.

Lemma 5 (Parallel composition). $(F, \otimes, \{(\langle \rangle, \langle \rangle)\})$ *is a (total) monoid, where multiplication is defined by* $((x, y) \otimes (a, b)) = (x \frown a, y \frown b)$.

The following corollary combines these two results.

Corollary 6. $(F, \gg, \otimes, id, \{(\langle \rangle, \langle \rangle)\})$ *is a partial bi-monoid.*

Note that because we view stream-processing functions as tuples of inputs to tuples of outputs, $f(x_1, x_2)$ may not have the same meaning as $f(x_2, x_1)$, i.e., the parallel composition operator is not necessarily commutative. Commutativity can be regained by using streams of type $T \to V \to X$, mapping variable names

V to values X. We leave the study of the (more complicated) stream processing functions that result from these as a topic of future study.

Clearly, it should be possible for two components operating in parallel to share inputs, or produce an output that combines the outputs of the two components. Such situations can be easily modelled by defining for instance, a duplicator that splits some shared input stream into two disjoint outputs. Similarly, outputs can be combined by a stream-processing function that collates, combines and processes outputs from several parallel sources. An example is given in Fig. 4, which defines the component $c_1 \gg (f \otimes g) \gg c_2$.

4 Feedback

The streams under consideration are over a linear order T. For such models, the use of Banach's theory to ensure the existence of a unique fixed point is well known [17, 4]. This includes constructive fixed-point theorems that enable calculation of this unique fixed point [4]. We recap Cataldo et al.'s main result (and the background needed to understand this result); then apply it to our setting of (m, n)-ary stream-processing functions.

Feedback algebraically. Following Cataldo et. al., the generalisation of Banach's fixed-point theory is given in terms of a *pomonoid* (as in *p*artially *o*rdered *monoid*), which is a structure $(\Gamma, \sqsubseteq, \oplus, \bot)$ such that $(\Gamma, \oplus, \bot)$ is a monoid and $(\Gamma, \sqsubseteq)$ is a partial order with minimum element $\bot$. Given a set X and a pomonoid $(\Gamma, \sqsubseteq, \oplus, \bot)$, we define a *petric* (as in *p*omonoid m*etric*) to be any $d : X \times X \to \Gamma$ such that for all $x, y, z \in X$:

1. $d\,x\,y = \bot$ iff $x = y$,
2. $d\,x\,y = d\,y\,x$, and
3. $d\,x\,z \sqsubseteq d\,x\,y \oplus d\,y\,z$

For example, any metric is a petric over the pomonoid $(R_{\geq 0}, \leq, +, 0)$.

An infinite sequence $G = (\gamma_0, \gamma_1, \dots) \in \Gamma^\omega$ is *decaying* iff for all $\gamma \in \Gamma \backslash \{\bot\}$ there exists an $n \in N$ such that for all $k \geq n$, $\gamma_k \sqsubset \gamma$, i.e., for any non-zero value γ, there is a point in G where the elements from that point onwards are below γ. An infinite sequence $X_s = (x_0, x_1, \dots) \in X^\omega$ is *Cauchy* iff for all $\gamma \in \Gamma \backslash \{\bot\}$, there exists an $n \in N$ such that for all $k, m \geq n$, $(d\,x_k\,x_m) \sqsubset \gamma$. We say that X_s *converges* to $x \in X$ iff the sequence $((d\,x_0\,x), (d\,x_1\,x), \dots) \in \Gamma^\omega$ is decaying. The set X is *Cauchy complete* iff for all Cauchy sequences $(x_0, x_1, \dots) \in X^\omega$, there exists a unique $x \in X$ such that the sequence $(x_0, x_1, \dots)$ converges to x.

These definitions are used to define a scheme for constructing the fixed point of a function $f : X \to X$, given by the following recursion, where $i \geq 0$:

$$f^0\,x = x \qquad\qquad f^{i+1}\,x = f\,(f^i\,x)$$

We say f is a *strict contraction* iff $\forall x, y \in X\ .\ x \neq y \Rightarrow d\,(f\,x)\,(f\,y) \sqsubset d\,x\,y$ for some petric d. For a discrete time domain, a *strict contraction* is enough to
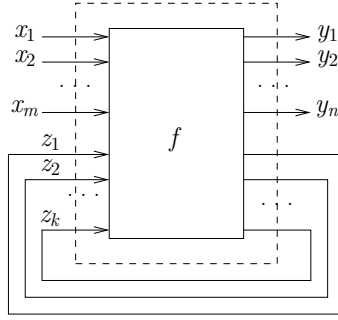
Fig. 5. Feedback composition $\mu^k f$

ensure a fixed-point is reached. Given $x, y \in X$ and $n \in \mathbb{N}$, let

$$B_n \, x \, y = \left\{ \bigoplus_{i=n}^{k} d\left(f^i \, x\right)\left(f^i \, y\right) \mid k \in \mathbb{N} \wedge k \geq n \right\}$$

A strict contraction f is a *decaying contraction* iff for all $x, y \in X$, there exists a decaying sequence $(\gamma_0, \gamma_1, \dots) \in \Gamma^\omega$ where γ_n is an upper bound for $B_n \, x \, y$.

Theorem 7 ([4]). *If X is Cauchy complete with respect to petric d, and if $f : X \to X$ is a decaying contraction, then f has a unique fixed point $fix(f) \in X$. Moreover, for any $x \in X$, the sequence $((f^0 \, x), (f^1 \, x), \dots)$ converges to $fix(f)$.*

Feedback for stream-processing functions. We now define feedback for stream-processing functions, which feeds k outputs of an $(m + k, n + k)$-ary delayed behaviour back to k inputs (see Fig. 5). Notation $\pi_{[i,j]}(x_1, x_2, \dots, x_n)$ denotes the projection $\pi_{[i,j]}(x_i, x_{i+1}, \dots, x_j)$ for $1 \leq i \leq j \leq n$.

Definition 8. *Let $f : X^{T,m} \times Z^{T,k} \to Y^{T,n} \times Z^{T,k}$ be an $(m + k, n + k)$-ary stream-processing function. Then $\mu^k f$ is a (m, n)-ary stream-processing function such that the value $(y_1, \dots, y_n)$ of $(\mu^k f)(x_1, \dots, x_m)$ is given by*

$$(y_1, \dots, y_n, z_1, \dots, z_k) = f\left(x_1, \dots, x_m, z_1, \dots, z_k\right)$$

where $(z_1, \dots, z_k)$ is the solution of the equation

$$(z_1, \dots, z_k) = \pi_{[n+1,n+k]} \, f\left(x_1, \dots, x_m, z_1, \dots, z_k\right) . \tag{1}$$

The theorem below follows immediately via an application of Cataldo et al's result for *eventually decaying* contractions. We elide the definition of eventually decaying, simply noting that every decaying contraction is eventually decaying.

Theorem 9. *If $f : X \to X$ is κ-causal, then f is a decaying contraction and has a unique fixed point.*

Corollary 10. *If $f : X^{T,m} \times Z^{T,k} \to Y^{T,n} \times Z^{T,k}$ is κ-causal, then $\pi_{[n+1,n+k]} \, f$ is a decaying contraction and has a unique fixed point.*

Example 11. Consider the controller in Example 2 operating in parallel with an environment (which modifies *temp*) depending on the value of the motor. We define

$$C_E(motor) = \lambda t : T \ . \ \text{if } motor \ t = on \text{ then } lower \ t \text{ else } raise \ t$$

where we assume *lower* (respectively, *raise*) is a continuous monotonically decreasing (increasing) function describing the rate of change of *temp*. The overall system is described by the composition: $\mu^1(C \gg C_E)$. This function is well-defined since its fixed point is uniquely determined. $C \gg C_E$ is contractive with delay κ, and hence, Corollary 10 can be applied.

5 Quantales and power series

The framework we have defined thus far enables reasoning about and composing stream-processing functions. We wish to extend this into a reasoning framework, and to this end, incorporate an interval temporal logic [10, 5, 16, 19], which may be used to reason about the safety, liveness, and real-time properties that a system possesses. It turns out that this extension can be constructed using an algebraic approach, by lifting the notion of a stream-processing function to a behaviour, which is a predicate over a stream-processing function and an interval.

This section presents the algebraic underpinnings to make the above aims possible. A *quantale* is a structure $(Q, \leq, \cdot, 1)$ such that $(Q, \leq)$ is a complete lattice, $(Q, \cdot, 1)$ is a monoid and the distributivity axioms

$$\left(\sum_{i \in I} x_i \right) \cdot y = \sum_{i \in I} (x_i \cdot y), \qquad x \cdot \left(\sum_{i \in I} y_i \right) = \sum_{i \in I} (x \cdot y_i)$$

hold, where $\sum X$ denotes the supremum of a set $X \subseteq Q$. We write 0 and U for the least and the greatest elements of the quantale with respect to $\leq$. The two annihilation laws $x \cdot 0 = 0 = 0 \cdot x$ hold in any quantale.

Example 12. The quantale of booleans $\mathbb{B} = \{0, 1\}$ with $0 \leq 1$, binary supremum or join $\sqcup$ and composition as binary infimum or meet $x \cdot y = x \sqcap y$ plays an important role for interval logics. It also satisfies distributivity laws with respect to join and meet and every element is complemented.

Convolution. The algebraic foundations for this paper is based on power series from formal languages, which provides mechanisms for lifting properties of the underlying algebraic structures to the level of functions over these structures. More formally, a *power series* is a function $f : M \to Q$ from a partial monoid M into a quantale Q. Operators on f are defined by lifting operators on M and Q as follows. For $f, g : M \to Q$, an index set I, a family of functions $f_i : M \to Q$ and $i \in I$, we define

$$\left(\sum_{i \in I} f_i \right) x = \sum_{i \in I} f_i \ x \qquad\qquad (f \cdot g) \ x = \sum_{x = y \circ z} (f \ y) \odot (g \ z)$$

Note that the first operation is just pointwise lifting with $(f + g)\, x = f\, x + g\, x$ as a special case. The composition $f \cdot g$ is called *convolution*. The variables y and z underneath the sum are implicitly existentially quantified. A more precise but less convenient notation is $(f \cdot g)\, x = \sum\{q \in Q \mid \exists y, z.\, x = y \circ z \wedge q = f\, y \odot g\, z\}$. The sum is lifted pointwise; $(f + g)\, x = f\, x + g\, x$ arises as a special case. In addition, we define the $\mathbb{0} : M \to Q$ and $\mathbb{1} : M \to Q$ by

$$\mathbb{0}\, x = 0, \qquad \mathbb{1}\, x = \text{if } x \in E \text{ then } 1 \text{ else } 0.$$

Hence $\mathbb{0}$ is the constant function that returns value 0 and $\mathbb{1}$ is the subobject classifier for E. The quantale structure lifts from Q to the function space Q^M of power series.

Theorem 13 ([8]). *Let $(M, \circ, D, E)$ be a partial monoid. If $(Q, \leq, \odot, 1)$ is a unital quantale, then so is $(Q^S, \leq, \cdot, \mathbb{1})$.*

The order $\leq$ on Q^M is obtained from that on Q by pointwise lifting: $f \leq g$ iff $f\, x \leq g\, x$ holds for all $x \in M$.

There are a variety of instantiations for quantale Q^M. Here, we are mainly interested in the quantale $\mathbb{B}^M \cong \mathcal{P}\, M$ of power series of type $M \to \mathbb{B}$ into the quantale of booleans, which is the power set quantale of the partial monoid M. In this instance, convolution becomes

$$(p \cdot q)\, x = \sum_{x = y \circ z} p\, y \sqcap q\, z \ .$$

Moreover, $\mathbb{1} = E$ is a boolean-valued function, hence $\mathbb{1}\, x$ holds iff $x \in E$. The boolean algebra structure of $\mathbb{B}$ is preserved by the lifting to $\mathbb{B}^M$. Hence distributive laws between join and meet hold and boolean complements of predicates can be defined.

As with monoids, it is possible to extend quantales with more than one multiplication operator. For example, a *bi-quantale* is a structure $(Q, \leq, \cdot_1, \cdot_2)$ such that $(Q, \leq, \cdot_1)$ and $(Q, \leq, \cdot_2)$ are quantales. A bi-quantale is *unital* iff both its multiplications have units.

6 Interval-stream specifications

With the necessary algebraic background in place, we develop our interval-based reasoning framework. The basis for this work is a specification construct that defines behaviours of system components using *interval-stream predicates*, which are predicates over an interval and an (m, n)-ary stream-processing function.

Formally, we assume $I(T) = \{[a, b] \mid a, b \in T \wedge a \leq b\}$ denotes the set of all (closed) intervals over the linear poset $(T, \leq)$. An interval-stream predicate has type $I(T) \times F \to \mathbb{B}$, mapping a given interval and stream-processing function to a boolean. Interval stream predicates can be understood as expressing properties of a stream-processing function f applied to an interval φ. They are similar to higher-order functions such as maps or folds in functional programming.

Example 14. Consider the specification of a system that controls a *motor* depending on the input value of the *temp*. Suppose we wish to specify that the *motor* is on at the end of any interval φ in which *temp* stays above K_{max}. This may be formalised by the interval-stream predicate *React*, where:

$$React \, \varphi \, (temp, motor) = (\forall t : \varphi \, . \, temp \, t > K_{max}) \Rightarrow motor \, (max \, \varphi) = on$$

Now recall the controller C from Example 2. Clearly, *React* φ (*temp*, C *temp*) does not necessarily hold because ϕ may refer to a time prior to system initialisation, or C may not have enough time to react within ϕ. However, it is possible to show that, for any φ such that $min \, \varphi \geq 0$ and $max \, \varphi - min \, \varphi > \kappa$, we have *React* φ (*temp*, C *temp*). □

Combining intervals and stream-processing functions algebraically. We develop an algebraic construction of interval-stream predicates using our convolution-based liftings. First, we must understand the algebraic structure of intervals. It is straightforward to show that intervals form a partial monoid. Let

$$D_{CI} = \{(a, b) \in I(T) \times I(T) \mid max \, a = min \, b\} \qquad E_{CI} = \{[t, t] \mid t \in T\}$$

be the domain of composition and set of all point intervals, respectively. Define the *interval fusion product* $a \; ; \; b = a \cup b$ that composes two intervals $[t_1, t_2]$ and $[u_1, u_2]$ by taking their union $[t_1, t_2]$ whenever $t_2 = u_1$. This turns $(I(T), ;, D_{CI}, E_{CI})$ into a partial monoid.

Note 15. An algebraic treatment of semi-open intervals can also be given [8], which leads to an alternative interval logic [5] that simplifies reasoning about discontinuities when discrete values change. However, because such a logic is more complex, we leave out this variation in this paper, and consider full development of such a framework to be future work.

Recall that we have already established that partial stream-processing functions form a bi-monoid (Corollary 6). Combining this result with the interval monoid results in a carrier set of type $\mathbf{M} = I(T) \times F$ and three partial multiplication operators:

- $;$ that operates as chop on the intervals;
- $\gg$ that operates as pipeline on the stream-processing functions; and
- $\otimes$ that operates as parallel composition on the stream-processing functions.

This results in a partial tri-monoid $(\mathbf{M}, ;, \gg, \otimes, E_;, E_{\gg}, E_{\otimes})$, where:

$$(z_1, f) \; ; \; (z_2, f) = (z_1 \; ; \; z_2, f)$$
$$(z, f_1) \gg (z, f_2) = (z, f_1 \gg f_2)$$
$$(z, f_1) \otimes (z, f_2) = (z, f_1 \otimes f_2)$$

define the three monoidal operations. The chop operates on the interval component, leaving the stream-processing function unchanged, while the pipeline and

parallel composition operators are applied to the functional component, leaving the interval component unchanged.

The definedness relation for the partial relations are given by lifting the definedness relations to the level of the cross product:

$$D_; = \{(x_1, f_1) \times (x_2, f_2) \mid (x_1, x_2) \in D_{CI} \wedge f_2 = f_2\}$$
$$D_\gg = \{(x_1, f_1) \times (x_2, f_2) \mid x_1 = x_2 \wedge (f_1, f_2) \in D_{OP}\}$$
$$D_\otimes = \{(x_1, f_1) \times (x_2, f_2) \mid x_1 = x_2\}$$

The unit sets for the three operators are $E_; = \{(i, f) \mid i \in E_{CI} \wedge f \in F\}$, $E_\gg = \{(i, f) \mid i \in I(T) \wedge f \in \mathrm{id}\}$ and $E_\otimes = \{(i, (\langle\rangle, \langle\rangle)) \mid i \in I(T)\}$.

Tri-quantales. Our aim is to lift these monoidal operations to the level of the interval-stream predicates using convolution. First we define the generic theory over the structure $Q^{M_1 \times M_2}$, where M_1 is a monoid, M_2 is a bimonoid and Q is quantale.

Theorem 16 below shows that this lifting gives us a tri-quantale structure in the generic case when the target algebra is a quantale. Later, we will instantiate this theorem and obtain our theory of interval predicates. Suppose $(M_1, \circ_1, E_1)$ is a partial monoid, and $(M_2, \circ_2, \circ_3, E_2, E_3)$ a partial bi-monoid. Define a structure

$$\mathcal{Q} = (Q^{M_1 \times M_2}, \leq, \cdot_1, \cdot_2, \cdot_3, \mathbb{1}_1, \mathbb{1}_2, \mathbb{1}_3)$$

where the three multiplication operators over $Q^{M_1 \times M_2}$ are defined using convolution as follows for $p, q \in Q^{M_1 \times M_2}$:

$$(p \cdot_1 q)(\varphi, f) = \sum_{\varphi = \varphi_1 \circ_1 \varphi_2} p(\varphi_1, f) \circ q(\varphi_2, f)$$

$$(p \cdot_2 q)(\varphi, f) = \sum_{f = f_1 \circ_2 f_2} p(\varphi, f_1) \circ q(\varphi, f_2)$$

$$(p \cdot_3 q)(\varphi, f) = \sum_{f = f_1 \circ_3 f_2} p(\varphi, f_1) \circ q(\varphi, f_2)$$

Theorem 16. *If $(M_1, \circ_1, E_1)$ is a partial monoid, $(M_2, \circ_2, \circ_3, E_2, E_3)$ is a partial bi-monoid and $(Q, \leq, \circ)$ is a unital quantale, then $\mathcal{Q}$ is a tri-quantale. Furthermore, if $(Q, \leq, \circ)$ is distributive, then so is $\mathcal{Q}$.*

As an example, we verify the unit law for the first multiplication operator.

$$(\mathbb{1}_1 \cdot_1 q)(\varphi, f)$$
$$= \sum_{\varphi = \varphi_1 \circ_1 \varphi_2} \mathbb{1}_1(\varphi_1, f) \circ q(\varphi_2, f)$$
$$= (\sum_{\substack{(\varphi = e \circ_1 \varphi) \\ e \in E_1}} \mathbb{1}_1(e, f) \circ q(\varphi, f)) + (\sum_{\substack{\varphi = \varphi_1 \circ_1 \varphi_2 \\ \varphi_1 \notin E_1}} \mathbb{1}_1(\varphi_1, f) \circ q(\varphi_2, f))$$
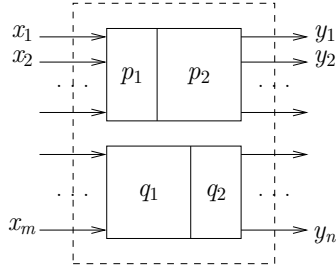
Fig. 6. $(p_1 \,;\, p_2) \otimes (q_1 \,;\, q_2)$

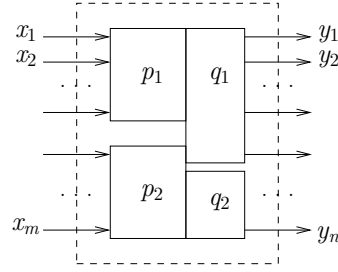**Fig. 7.** $(p_1 \otimes p_2) \,;\, (q_1 \otimes q_2)$

$$= (\sum_{\substack{(\varphi = e \circ_1 \varphi) \\ e \in E_1}} \top \circ q\,(\varphi, f)) + (\sum_{\substack{\varphi = \varphi_1 \circ_1 \varphi_2 \\ \varphi_1 \notin E_1}} 0 \circ q\,(\varphi_2, f))$$

$$= (\top \circ q\,(\varphi, f)) + 0$$

$$= q\,(\varphi, f).$$

Power series over $\mathbf{M}$. To apply Theorem 16 to our setting of interval-stream predicates, we instantiate the monoidal structure to $\mathbf{M}$ and the quantale to the boolean quantale $\mathbb{B}$. Thus we obtain the following corollary.

Corollary 17. $(\mathbb{B}^{\mathbf{M}}, \leq, \,;\,, \gg, \otimes, \mathbb{1}_{;}, \mathbb{1}_{\gg}, \mathbb{1}_{\otimes})$ *is a unital distributive tri-quantale.*

Although these operators have a similar algebraic structure, they manipulate their arguments in different ways, which highlights the uniformity and power of our approach. The predicate $p \,;\, q$ holds for a function f and interval $[a, b]$, if that interval can be split into two subintervals $[a, c]$ and $[c, b]$ such that p holds for f and $[a, c]$ and q holds for f and $[c, b]$. Predicate $p \gg q$ holds for a function f and interval φ if f consists of the composition f_1 of f_2 such that p holds for f_1 and φ and q holds for f_2 and φ. Predicate $p \otimes q$ is similar to $p \gg q$, except f must be split using $\otimes$.

The differences are most apparent when we consider interval-stream predicates containing combinations of these operations. For instance, consider the differences between $(p_1 \,;\, p_2) \otimes (q_1 \,;\, q_2)$ and $(p_1 \otimes p_2) \,;\, (q_1 \otimes q_2)$, which are depicted in Figs. 6 and 7, respectively In Fig. 6, the initial component is first split into two parallel subcomponents, then, using $;$, the intervals in which these subcomponents operate are split. Note that the two splittings of the intervals are independent, because the parallel composition guarantees this. On the other hand, in Fig. 7, the interval split occurs first, and for each of the subintervals, the parallel composition operator splits the stream functions into two disjoint subsets.

It is possible to perform a similar exercise using $\gg$ in place of $\otimes$, i.e., consider the difference between $(p_1 \,;\, p_2) \gg (q_1 \,;\, q_2)$ and $(p_1 \gg p_2) \,;\, (q_1 \gg q_2)$, as depicted in Figs. 8 and 9, respectively. In Fig. 8, the initial component is first split using pipelined composition, which requires that we find a set of outputs of $(p_1 \,;\, p_2)$
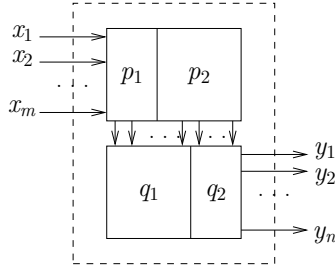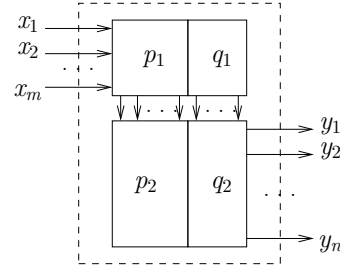
Fig. 8. $(p_1 \; ; \; p_2) \gg (q_1 \; ; \; q_2)$ **Fig. 9.** $(p_1 \gg p_2) \; ; \; (q_1 \gg q_2)$

that can be used as inputs to $(q_1 \; ; \; q_2)$. The intervals arguments to $p_1 \; ; \; p_2$ and $q_1 \; ; \; q_2$ can be split independently. On the other hand, in Fig. 9, the interval split occurs first, and for each of these subintervals, it must be possible to find a intermediate set of outputs of p_i that can be used as inputs to q_i.

7 Modalities over interval-stream predicates

We have extended a functional specification framework with intervals. Modal (and temporal) logics for intervals are well studied. In this section, we show how these existing works can be extended to cope with modal (temporal) reasoning over functional specifications. In addition, by exploiting the uniformity of our (convolution-based) algebraic construction, we develop a novel method for reasoning over compositions of functional specifications by adapting interval modalities.

A negation operator $\neg$ is available for every boolean quantale, which can be lifted point-wise to the level of our interval-stream tri-quantale $\mathbb{B}^{\mathbf{M}}$. The chop operator can be used to define eventually p ($\Diamond p$) and combined with $\neg$ to define ($\Box p$) as follows:

$$\Diamond p = \top \; ; \; p \; ; \; \top \qquad\qquad \Box p = \neg \Diamond \neg p$$

Thus $(\Diamond p)\,(\varphi, f)$ holds iff the interval component there is some subinterval of φ' of φ such that $p\,(\varphi', f)$ holds. In other words, if $\varphi = [a, b]$, then $(\Diamond p)\,(\varphi, f)$ holds iff $p\,([a', b'], f)$ where $a \leq a' \leq b' \leq b$. On the other hand, $(\Box p)\,(\varphi, f)$ holds iff $p\,(\varphi', f)$ holds for every subinterval φ' of φ.

Note 18. The definition for $\Diamond p$ must be modified if infinite intervals are considered. Namely, the first $\top$ within $\Diamond p$ must be replaced by an element *fin*, which is a predicate that returns $\top$ iff the given interval is finite. For an algebraic treatment, see for example [8, 14].

The example below shows how one can use these modalities to develop specifications as predicates over interval stream-processing functions.

Example 19. Suppose we wish to specify a component f that satisfies the property for an input interval φ:

"if the input temperature $temp$ is ever above K_{max} for k time units, then the output $motor$ is set to on sometime within φ".

We construct the interval-stream predicate bottom up to demonstrate how the logic works. First we define a predicate for the first part of the antecedent:

$$\mathsf{higher}\,\varphi\,(temp, motor) = (max\,\varphi - min\,\varphi \geq k) \wedge (\forall t \in \varphi\,.\,(temp\,t) > K_{max})$$

The first conjunct states that the length of φ is at least k and the second states that the value $temp$ within for each time t in φ is above K_{max}. Note that the output component $motor$ is ignored on the right hand side of the equation above, but is present to enable the functions below to be defined using lifting constructs. We are now able to express the property that the temperature eventually rises above K_{max} using the $\diamond$ operator:

$$\mathsf{ev\_higher}\,\varphi\,(temp, motor) = (\diamond\mathsf{higher})\,\varphi\,(temp, motor)$$

Thus $\mathsf{ev\_higher}\,\varphi\,(temp, motor)$ holds iff there is some subinterval φ' of φ such that $\mathsf{higher}\,\varphi'\,(temp, motor)$ holds. In particular, $\diamond$ is defined in terms of $;$, which only splits the interval argument. Next, we define an interval-stream predicate for the consequent:

$$\mathsf{motor\_on}\,\varphi\,(temp, motor) = \exists t : \varphi\,.\,(motor\,t) = on$$

With this, we arrive at an interval-stream predicate that formalises the requirement above:

$$\mathsf{Spec} = \mathsf{ev\_higher} \Rightarrow \mathsf{motor\_on}$$

Returning to our component C from Example 2, it is straightforward to show $\mathsf{Spec}\,\varphi\,(temp, C\,temp)$ holds for any interval φ such that $min\,\varphi \geq 0$.

Modalities over stream-processing functions. The modalities over intervals as defined above are standard; the difference here is that they are applied to stream-processing functions. Our algebraic construction highlights the structural similarities between chop $;$ defined for intervals, and pipeline $\gg$ and parallel $\otimes$ composition defined for stream-processing functions, which provides us with an opportunity to define new modalities over the input/output pairs. In particular, we define modalities analogous to $\diamond$ as follows:

$$\diamond_{\gg} p = \top \gg p \gg \top \qquad\qquad \Box_{\gg} p = \neg\diamond_{\gg}\neg p$$
$$\diamond_{\otimes} p = \top \otimes p \otimes \top \qquad\qquad \Box_{\otimes} p = \neg\diamond_{\otimes}\neg p$$

Thus $(\diamond_{\gg} p)\,\varphi\,f$ holds iff f is of the form $f_1 \gg f_2 \gg f_3$ such that $p\,\varphi\,f_2$ holds. Similarly, $(\diamond_{\otimes} p)\,\varphi\,f$ holds iff f is of the form $f_1 \otimes f_2 \otimes f_3$ and $p\,\varphi\,f_2$ holds. Both operators $\diamond_{\gg}$ and $\diamond_{\otimes}$ are useful for stating the existence of a subcomponent that satisfies property p over the given interval φ. Dually, $\Box_{\gg} p$ iff for any pipelined decomposition p holds for that decomposition ($\Box_{\otimes}$ is similar). We leave full development of such a theory as future work.

8   Conclusion and future work

We have algebraically constructed a logic for reasoning about stream-based systems. Applying these constructions to hybrid systems, we obtain a flexible computation model, in contrast to existing model-theoretic approaches [15, 11, 18] that are defined using automata (or similar transition-system-like model), which are somewhat rigid in their structure. Our constructions unify reasoning whenever possible; the theoretical underpinnings are provided by convolution [8], which enables operators to be lifted to the level of functions. Our work is distinguished from other algebras for hybrid systems [14, 8, 7], which do not distinguish between inputs and outputs using stream-processing functions.

This work is still in its initial stages, but presents a method for bringing algebraic reasoning into hybrid systems [8]. Areas such as network theory have already benefitted from the generality, conciseness and uniformity that algebraic reasoning enables [1]. Future work will include development of neighbourhood logics [10, 13], Hoare logics [8] and mechanisation [6]. Due to the quantale-like structure of our algebra, the mathematical foundations are already available, and hence, these planned future works can be rapidly developed.

References

1. C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: semantic foundations for networks. In *POPL*, pages 113–126. ACM, 2014.
2. J. Berstel and C. Reutenauer. *Les Séries Rationnelles et Leurs Langagues*. Masson, 1984.
3. M. Broy. Refinement of time. *Theor. Comput. Sci.*, 253(1):3–26, 2001.
4. A. Cataldo, E. Lee, Xiaojun Liu, E. Matsikoudis, and Haiyang Zheng. A constructive fixed-point theorem and the feedback semantics of timed systems. In *Discrete Event Systems*, pages 27–32, July 2006.
5. B. Dongol and J. Derrick. Interval-based data refinement: A uniform approach to true concurrency in discrete and real-time systems. *Sci. Comput. Program.*, 111:214–247, 2015.
6. B. Dongol, V. B. F. Gomes, and G. Struth. A program construction and verification tool for separation logic. In R. Hinze and J. Voigtländer, editors, *MPC*, volume 9129 of *LNCS*, pages 137–158. Springer, 2015.
7. B. Dongol, I. J. Hayes, L. Meinicke, and K. Solin. Towards an algebra for real-time programs. In W. Kahl and T. G. Griffin, editors, *RAMiCS*, volume 7560 of *LNCS*, pages 50–65. Springer, 2012.
8. B. Dongol, I. J. Hayes, and G. Struth. Convolution as a unifying concept: Applications in separation logic, interval calculi, and concurrency. *ACM Trans. Comput. Log.*, 17(3):15, 2016.

9. M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. Springer, 2009.

10. V. Goranko, A. Montanari, and G. Sciavicco. A road map of interval temporal logics and duration calculi. *Journal of Applied Non-Classical Logics*, 14(1-2):9–54, 2004.

11. T. A. Henzinger. The theory of hybrid automata. In *LICS '96*, pages 278–292, Washington, DC, USA, 1996. IEEE Computer Society.

12. T. Hoare and S. van Staden. In praise of algebra. *Formal Asp. Comput.*, 24(4-6):423–431, 2012.

13. P. Höfner and B. Möller. Algebraic neighbourhood logic. *J. Log. Algebr. Program.*, 76(1):35–59, 2008.

14. P. Höfner and B. Möller. An algebra of hybrid systems. *J. Log. Algebr. Program.*, 78(2):74–97, 2009.

15. N. Lynch, R. Segala, and F. Vaandraager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

16. B. C. Moszkowski. A complete axiomatization of interval temporal logic with infinite time. In *LICS*, pages 241–252, 2000.

17. O. Müller and P. Scholz. Functional specification of real-time and hybrid systems. In O. Maler, editor, *HART*, volume 1201 of *LNCS*, pages 273–285. Springer, 1997.

18. M. Rönkkö, A. P. Ravn, and K. Sere. Hybrid action systems. *Theor. Comput. Sci.*, 290:937–973, January 2003.

19. C. Zhou and M. R. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. EATCS: Monographs in Theoretical Computer Science. Springer, 2004.