Signal Rate Inference for Multidimensional Faust

Yann Orlarey
Grame, France
orlarey@grame.fr

Pierre Jouvelot
MINES ParisTech, PSL Research University, France
pierre.jouvelot@mines-paristech.fr

ABSTRACT

We introduce a new signal-level, type- and rate-based semantic framework for describing a multirate version of the functional, domain-specific Faust language, dedicated to audio signal processing, and here extended to support array-valued samples. If Faust is usually viewed as a formalism for combining *signal processors*, which are expressions mapping input signals to output signals, we provide here the first formal, lower-level semantics for Faust based on *signals* instead. In addition to its interest in understanding the inner workings of the Faust compiler, which uses symbolic evaluation of signal expressions, this approach turns out to be useful when introducing a language extension targeting multirate and multidimensional (array-valued) processing.

More precisely, we provide (1) new syntax and dynamic semantics for (recursive) Faust-based signals, (2) a type and, more interestingly, rational rate static semantics and (3) a new rate inference algorithm, together with its soundness and (relative) completeness theorems.

CCS CONCEPTS

•Theory of computation →Streaming models; Type theory; •Applied computing →Sound and music computing; •Software and its engineering →Domain specific languages;

KEYWORDS

Faust, audio signal processing, type systems, rate inference

ACM Reference format:
Yann Orlarey and Pierre Jouvelot. 2016. Signal Rate Inference for Multidimensional Faust. In *Proceedings of IFL Conference, Leuven, Belgium, August 31-September 02, 2016 (IFL 2016)*, 12 pages.
DOI: http://dx.doi.org/10.1145/3064899.3064902

1 INTRODUCTION

The specifics of computer music call for the design and implementation of domain-specific programming languages (DSL) [1]. Faust is one of these languages for real-time signal processing applications, in particular real-time audio processing [18]; it boasts a thriving community of users, both in academia (e.g., [19]) and industry[1]. Faust is based on a few core foundational principles.

Real-time signal processing. Faust is aimed at the specification and efficient implementation of real-time programs based on causal computations, with bounded memory and CPU footprints, and minimal latency.

Simple well-defined formal semantics. Faust does not intend to model the internal behavior of systems or circuits. The only "interesting" semantics is the one that can be observed from outside, i.e., a function that maps a tuple of time-dependent input signals to a tuple of output signals.

High-level specification. Faust is designed to be a high-level *specification language* rather than an implementation language. A key design choice is to make a clear separation between the users' role, in charge of specifications, and the role of the compiler, in charge of implementing them. The way the user writes a Faust program should not matter; only its meaning should count. Ideally two different Faust programs with the same meaning should have the same implementation[2]; this allows for two syntactically different but semantically equivalent code fragments to be sometimes compiled to the same code (computer music programmers often use cut-and-paste operations, in particular during "live coding" sessions), while some programs that are somewhat convoluted to write in Faust may be seen, at a lower level, as more simple expressions on signals, and thus be efficiently compiled as such.

Functional approach. The functional paradigm that Faust adopts provides a high level of modularity, which eases both composition and understanding. Moreover, it offers a very natural framework for signal processing. Periodically-sampled digital signals can be modeled as functions of time. *Signal processors*, which are the primary constituants of Faust, are second-order functions operating on signals. Faust block-diagram algebra is a set of third-order composition operations on signal processors. Finally, user-defined functions can be seen as higher-order functions on block-diagram expressions.

The current version of Faust is monorate: all signals are isomorphic to functions mappings integers (clock ticks) to (scalar) sample values. In [12], an extension to Faust for handling both different clocks and multidimensional samples has been proposed: these features are of key importance when targeting efficient spectral processing applications. These clocks are introduced here as rational *rates* that interact with the size of the array-valued samples. When building a signal of vectors of size n from a signal of rate r carrying scalar samples, integer say, one gets a signal operating at rate r/n; conversely, serializing a signal of rate r carrying samples that are vectors of size n, the resulting signal has a rate rn. The overall purpose of this paper is to describe how this approach

[1] See, for instance, http://faustone.com.

[2] Such a requirement is obviously undecidable in general, but this does not preclude the Faust compiler from making its best effort to attain it.

IFL 2016, Leuven, Belgium

DOI: http://dx.doi.org/10.1145/3064899.3064902

can be handled within the Faust compiler infrastructure, while maintaining the general design principles sketched above.

To fulfill its goals, the Faust compiler uses optimization techniques based on a blend of symbolic evaluation and abstract interpretation approaches. Instead of using Faust signal processors directly as its core data structure to compile user code, it uses an intermediate representation (Faust IR) based on signal expressions[3]. Basically, a Faust signal processor is first converted to a tuple of signal expressions during a phase of symbolic propagation performed in the compiler front-end. Our rate inference algorithm operates directly on Faust IR expressions.

In this paper, we provide the following contributions:

- the first formal definition of Faust IR, including its extension to handle the new multirate framework of [12];
- a new definition of multirate signals, based on a rational model for its clocking mechanism;
- a new rate inference algorithm, for which both soundness and (relative) completeness theorems are specified and proven correct;
- a prototype implementation of this algorithm in an experimental multirate version of Faust.

In Section 2, we describe a proposal for multirate signals that use rational clocks. A multirate Faust IR based on typed and rated signals is introduced in Section 3, together with a clocked semantics and a Rate Subject Reduction property. We provide a set of type and rating rules adapted to multirate signals in Section 4 for which we state a (value) Subject Reduction property. The core of the paper is Section 5 where we describe our new rate inference algorithm, together with its soundness and (relative) completeness theorems. We briefly report on the related work in Section 6 and discuss possible future work in Section 7 before concluding.

2 MULTIRATE AND MULTIDIMENSIONAL SIGNALS

Here we are interested in periodically-sampled and multidimensional signals. We consider sampled signals as approximations of continuous signals, and we want to express signals sampled at various rates, but also signals with multidimensional sample values (that is not only signals of numbers, but also signals of fixed-size vectors of numbers, fixed-size vectors of fixed-size vectors of numbers, etc.).

We define below more precisely the notions of *time*, *sample value* and *signal* we are interested in.

2.1 Periodic time domain

In order to capture the idea of a sampled signal with a specific sampling rate, we introduce the concept of periodic time domain, notated $\mathbb{T}_r$. The idea is to "sample" the continuous time domain $\mathbb{R}$ with a periodicity represented by a rational rate r.

Definition 2.1 (Periodic time domain). The periodic time domain $\mathbb{T}_r$ is the set of rational values corresponding to the r-sampling of the continuous time domain $\mathbb{R}$, i.e.,:

$$\mathbb{T}_r = \frac{1}{r}\mathbb{Z} = \{\frac{i}{r} \mid i \in \mathbb{Z}\}, r \in \mathbb{Q}^* \ .$$

Here are some examples of time domains:

$$\mathbb{T}_1 = \{\dots, -2, -1, 0, 1, 2, \dots\} \ ;$$
$$\mathbb{T}_2 = \{\dots, -1, -0.5, 0, 0.5, 1, \dots\} \ ;$$
$$\mathbb{T}_{1/3} = \{\dots, -6, -3, 0, 3, 6, \dots\} \ .$$

Time domains have the following properties, for all $r \in \mathbb{Q}^*$ and $n \in \mathbb{N}^*$:

$$\mathbb{T}_r = \mathbb{T}_{-r} \ ;$$
$$\mathbb{T}_r \subseteq \mathbb{T}_{nr} \ ;$$
$$0 \in \mathbb{T}_r \ .$$

2.2 Signal

We can define a *multirate*, *multidimensional* signal as a function from a periodic time domain $\mathbb{T}_r$ to a set of multidimensional sample values V extended with a distinguished *zero* value noted 0_{V} (see Section 2.2.1 for an explanation).

Definition 2.2 (Signal). A multirate, multidimensional signal s is a function of time, from a time domain $\mathbb{T}_r$ to a set of multidimensional sample values V or 0_{V}:

$$s : \mathbb{T}_r \to \mathsf{V} \cup \{0_{\mathsf{V}}\} \ .$$

Definition 2.3. To simplify the notation of signals, we use the following abbreviation:

$$\mathsf{V}^r = \mathbb{T}_r \to \mathsf{V} \cup \{0_{\mathsf{V}}\} \ .$$

Sample values in V can be numbers (integers or floating-points) or fixed-size vectors of samples. These values are structurally typed, with a type in a domain T (see Section 4). Moreover numbers can be restricted to belong to an interval $[l, h]$, where l and h are either integers or reals depending of the considered type.

2.2.1 Negative time. The values of signals are usually needed starting from time 0 and up. But to take into account *delay operations*, negative times are always mapped to zeros. In operational terms, this corresponds to assuming that all delay lines are signals initialized with 0s.

Definition 2.4 (Negative time). The value of a signal $s : \mathsf{V}^r$ is always 0_{V} when $t < 0$:

$$\forall t \in \mathbb{T}_r, \ t < 0 \implies s(t) = 0_{\mathsf{V}} \ .$$

2.2.2 Constant signal. Because of Definition 2.4, a constant signal is usually not constant on its whole time domain, but only on its positive half. For example, for the constant integer signal 1 and $t \in \mathbb{Z} \geq 0$, we have $1(t) = 1$, but, if $t < 0$, one has $1(t) = 0_{\mathbb{N}}$ (see Figure 1).

Definition 2.5 (Constant signal). A signal $s : \mathsf{V}^r$ is constant iff

$$\forall t_1, t_2 \in \mathbb{T}_r, \ t_1 \geq 0 \text{ and } t_2 \geq 0 \implies s(t_1) = s(t_2) \ .$$

Note that the rate of constant signals is chosen to always be 1; this design decision ensures the referential transparency of Faust at the signal level, and we discuss possible variants in Section 7.

[3] In fact, the Faust compiler uses various IRs; we concentrate here on the signal-level Faust IR.

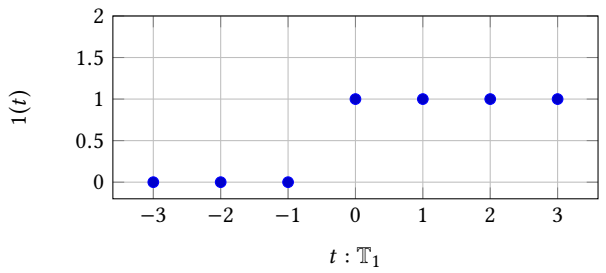


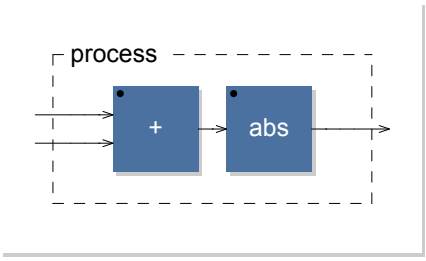
Figure 1: Constant signal 1

3 SIGNAL EXPRESSIONS

This section introduces the language of signal expressions that form the basis of Faust IR. We describe its syntax and semantics, with its multirate and multidimensional traits.

3.1 Introduction

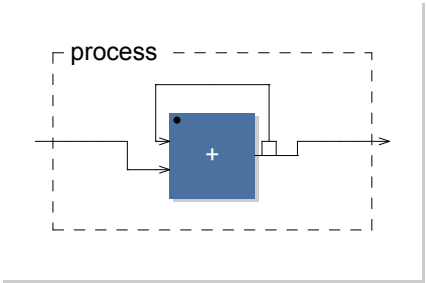
We use simple examples to illustrate how Faust expressions are encoded in terms of signal expressions. For instance, the Faust `process` below




```
process = + : abs ;
```

expects two (the arity of +) inputs, sums them and feeds the result (via the combinator :) to the absolute value signal processor. This Faust signal processor expression is converted into the Faust IR signal expression $\langle \text{abs}(\text{I}_0 + \text{I}_1)\rangle$ after propagating the tuple of input signals $\langle \text{I}_0, \text{I}_1\rangle$; as explained below, input signals are members of the dedicated domain I of symbolic identifiers.

New tuples $X = \langle X_0, ..., X_n\rangle$ of signal identifiers X_i are introduced, together with their definitions $D(X)$ as tuples of signal expressions, when ~-recursive expressions occur in Faust. For instance, the signal processor




```
process = + ~ _ ;
```

both outputs a signal S and feeds back (via the _ identity signal processor) this same S, after a one-sample delay[4], as the first argument of the + signal processor. This signal processor expression is converted to the signal expression X_0, together with a binding $D(X) = \langle X_0@1 + \text{I}_0\rangle$ where[5] $X = \langle X_0\rangle$, after propagating the tuple of input signals $\langle \text{I}_0\rangle$; there, the $@n$ expression denotes a n-sample delay operation. A more complex example, introducing the straighforward vector features proposed in [12], is given, in a graphical representation actually generated by the Faust compiler, in Figure 2.

3.2 Syntax

Definition 3.1 (Signal expression). A signal expression (E, D) in S, resulting from the phase of symbolic propagation, is defined by the following abstract syntax:

$$\begin{aligned} E \in S ::= {} & k \mid f \mid \text{I}_n \mid X_i \\ & \mid E_1 \star E_2 \mid E_1@E_2 \mid E \uparrow^n \mid E \downarrow_n \\ & \mid \mathbf{v}(E, n) \mid \mathbf{s}(E) \mid E_1 \# E_2 \mid E_1[E_2] \end{aligned} \tag{1}$$

where recursively-defined tuple lists of identifiers $X = \langle X_0, ..., X_{n-1}\rangle$ are bound in D, such that:

$$D(X) = \langle E_0, ..., E_{n-1}\rangle \tag{2}$$

for a number $n \in \mathbb{N}^+$ of recursively-defining expressions E_i.

In the previous definition, we assume that:

- k is a constant integer signal;
- f is a constant float signal;
- I_n represents an external input signal;
- X_i, with $D(X) = \langle E_0, ..., E_{n-1}\rangle$, represents the signal E_i of a group of mutually recursive signals;
- $E_1 \star E_2$ is a generic numerical operation on two signals;
- $E_1@E_2$ is a variable delay operation (the delay E_2 is measured in time steps);
- $E \uparrow^n$ is an up-sampling by a (strictly) positive factor $n \in \mathbb{N}^+$;
- $E \downarrow_n$ is a down-sampling by a factor $n \in \mathbb{N}^+$;
- $\mathbf{v}(E, n)$ is a vectorization operation of size $n \in \mathbb{N}^+$;
- $\mathbf{s}(E)$ is a serialization of a vector signal;
- $E_1 \# E_2$ is a concatenation of two vectors;
- $E_1[E_2]$ is an access to the element of index E_2 of the vector-valued signal E_1.

As can be noticed, even though signal expressions can be recursive, the language S is in fact closer to Kleene's primitive recursive functions [13] or recurrence equations than to a standard functional programming language. In fact, Faust functional status is mostly embedded in its higher-level, macro subsystem; all higher-order expressions expressed there are evaluated, at compile time, to generate signal expressions in S, which are studied in this paper.

3.3 Semantics

We define the semantics of a signal expression E with recursive definitions D (we omit D when not needed in the equations below) in an input environment a, mapping input identifiers to incoming

[4] This delay can be seen as a guarded condition that ensures causality (see for instance [8] for a general presentation of this issue).
[5] Here, X is a tuple, since there may be multiple recursive signals in a single recursive signal processor expression.

signals, and at a rate r as a function from time ticks in $\mathbb{T}_r$ to values[6] in some $\mathsf{V}'_{\perp_{\mathsf{V}'}}$, with $\mathsf{V}' = \mathsf{V} \cup \{0_{\mathsf{V}}\}$. As mentioned above, all signal samples for negative times in $\mathbb{T}_r$ have the appropriate 0_{V} value.

The denotational semantics of a signal expression (E, D) based on rational clocks is defined as

$$\lambda r.\lambda t.(\mathbb{S}^r_a(E)t, \text{ if } t \in \mathbb{T}_r, \text{ and } \perp_{\mathsf{V}}, \text{ otherwise}) ,$$

where $\mathbb{S}$ is (partially) defined as the least fixed point of the equations

$$\begin{aligned}
\mathbb{S}^r_a(k)t &= k, \text{ if } r = 1, \text{ and } \perp_{\mathsf{V}}, \text{ otherwise} ,\\
\mathbb{S}^r_a(f)t &= f, \text{ if } r = 1, \text{ and } \perp_{\mathsf{V}}, \text{ otherwise} ,\\
\mathbb{S}^r_a(I_n)t &= a(I_n)t ,\\
\mathbb{S}^r_a(X_i, D)t &= \mathbb{S}^r_a(\pi_i(D(X)))t ,\\
\mathbb{S}^r_a(E_1 \star E_2)t &= s^r_1(t) \star s^r_2(t) ,\\
\mathbb{S}^r_a(E_1@E_2)t &= s^r_1(t - s^r_2(t)/r) ,
\end{aligned}$$

where we note $s^r_i(t) = \mathbb{S}^r_a(E_i)t$ the value of the signal corresponding to E_i at Time t and Rate r, while π_i is the i-th projection operator on lists. As usual, we identify constant expressions, or those in normal form, with the values they denote. The multidimensional features are defined as

$$\begin{aligned}
\mathbb{S}^r_a(E_1 \uparrow^n)t &= s^{r_1}_1(\lfloor tr_1 \rfloor / r_1), \text{with } r_1 = r/n ,\\
\mathbb{S}^r_a(\mathbf{s}(E_1))t &= s^{r_1}_1(\lfloor tr_1 \rfloor / r_1)[\mathrm{mod}(rt, n)],\\
&\quad \text{with } |s^{r_1}_1(t)| = n \text{ and } r_1 = r/n ,\\
\mathbb{S}^r_a(E_1 \downarrow_n)t &= s^{rn}_1(t) ,\\
\mathbb{S}^r_a(\mathbf{v}(E_1, n))t &= [s^{r_1}_1(t - \delta_{n-1}), ..., s^{r_1}_1(t - \delta_0)] ,\\
&\quad \text{with } r_1 = nr \text{ and } \delta_j = j/r_1,\\
\mathbb{S}^r_a(E_1 \# E_2)t &= s^r_1(t) \# s^r_2(t) ,\\
\mathbb{S}^r_a(E_1[E_2])t &= s^r_1(t)[s^r_2(t)] .
\end{aligned}$$

where $|v|$ denotes the length of vector v.

3.4 Discussion

There are a couple of unusual features in the semantics we just introduced. First, note that the recursive signals as defined here are non-causal: nothing prevents meaningless expressions (E, D) such as $(\langle X_0 \rangle, \perp[X \to \langle X_0 + 1 \rangle])$, where $\perp$ denotes here a mapping for recursive definitions with an empty domain. In practice, the Faust compiler always adds an explicit 1-sample delay in recursive definitions, yielding expressions such as $\langle X_0 \rangle, \perp[X \to \langle (X_0@1) + 1 \rangle])$. Figure 2 provides an example of the introduction of these explicit delays on feedbacks, on a different example. In the sequel, we will always assume that signal expressions are syntactically causal, along the scheme just presented.

Second, the creation of vectors via the vectorize v construct is designed to minimize latency, a key issue in audio processing. In the multirate version of Faust, compiled to the signal language presented here, as soon as a single input sample is available, it is padded with 0s and an almost 0-filled vector is output. The vectorize construct semantics is illustrated in Figure 3 (some information regarding a possible implementation of vector operations based on local vector buffers V is provided in Figure 4). This unusual initialization process is designed to ensure the following property.

[6] As usual, a partially-ordered domain $X_{\perp_X}$ is defined for every set X, such that $\perp_X \sqsubset x$ for all $x \in X$. A partial function f defined over X and with values in Y can thus be seen as a total function from X to $Y_{\perp_Y}$; its domain of definition $\mathrm{Dom}(X)$ is defined as $\{x \in X \mid f(x) \neq \perp_Y\}$.

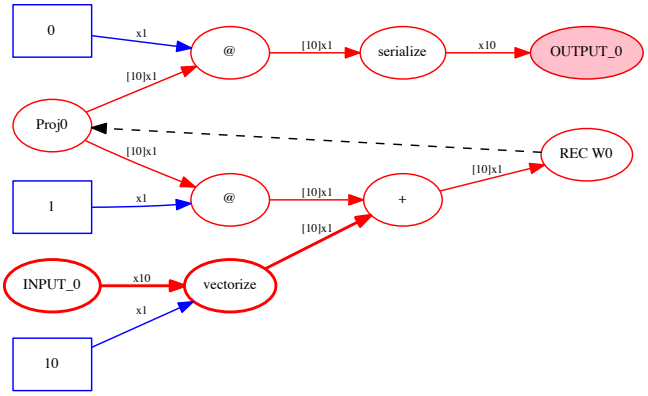


Figure 2: `process = vectorize(10,_) : +~_ : serialize;`

Figure 3: Vectorization by 3

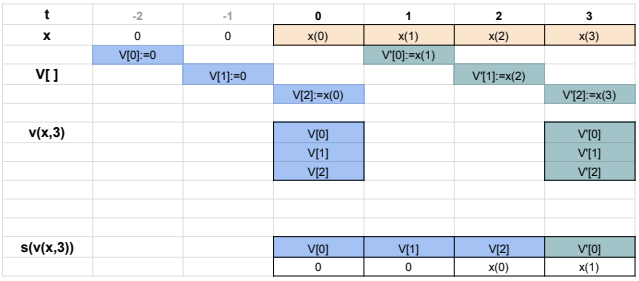


Figure 4: Vector run-time pipeline example (start)

Proposition 3.2 (Vector Inverse Property). *The signal semantics ensures that* $\mathbb{S}^r_a(\mathbf{s}(\mathbf{v}(E, n))) = \mathbb{S}^r_a(E@(n-1))$.

Finally, the upsampling operation does not introduce 0-padding as is often done in the literature. As described above, our model keeps the sample value constant between each time tick. Informally, we view the up- and down-sampling operations as performing "focus-varying" approximations over an (ideal) continuous function representing the "actual" signal, viewed as the limit of a family of step functions, i.e., constant over intervals. Performing upsampling

this way has the advantage, as we will see, of keeping the upsampled types tight, without having to perform a least upper bound operation with a zero type to accommodate for 0-padding.

4 TYPING RULES

Types and rates of signal expressions are defined by a set of rules $\Gamma \vdash (E, D) : \mathsf{T}^r$ indicating how to compute the type T^r of a signal expression E with recursive definitions D according to a signal type environment Γ storing type hypotheses for each input and recursive signal.

4.1 Types

Definition 4.1 (Sample types). The type domain T for sample values is defined recursively by the following syntax:

$$\mathsf{T} ::= \mathsf{int}[l,h] \mid \mathsf{float}[l,h] \mid [n]\mathsf{T} \,,$$

where types denote the set of values they abstract. Note how scalar types are annotated with a value interval (with integers or floats l and h). This information is crucial to ensure that Faust expressions can be compiled with a statically-known memory footprint, in particular when handling delay lines. This information is also used to ensure that vector operations are safe: the size information used to create them is known at compile time, while array accesses are known to be within array bounds.

Signals are associated to types derived from the type of their sample values. We note T^r the type of a signal that maps the time domain $\mathbb{T}_r$ to sample values of type T. This way, a signal in V^r has type T^r.

Definition 4.2 (Type model). A sample type T is associated to the set of values $\mathbb{M}(\mathsf{T})$ it denotes, defined inductively as:

- $\{n \in \mathbb{Z} \mid l \le n \le h\}$, if $\mathsf{T} = \mathsf{int}[l,h]$;
- $\{r \in \mathbb{R} \mid l \le r \le h\}$, if $\mathsf{T} = \mathsf{float}[l,h]$;
- $\{(v_0, ..., v_n) \mid v_i \in \mathbb{M}(\mathsf{T}_1)\}$, if $\mathsf{T} = [n]\mathsf{T}_1$, i.e., the set of fixed-size vectors of $n > 0$ elements in $\mathbb{M}(\mathsf{T}_1)$.

Note that our interval calculus does not allow for undefined values to occur. It is customary in other languages to assume that a type does not preclude the appearance of undefined values (denoted by the $\perp_{\mathsf{V}}$ symbol in the language semantics above). Our type system is designed to ensure, at compile time, that such "values" cannot occur in an actual well-typed signal expression; the function it denotes is always total. For instance, a signal expression trying to perform a division operation with a signal of divisors of sample type $\mathsf{int}[-2,2]$ will be rejected by the type checker[7]. Our interval calculus is also used to ensure that audio delay lines have a bounded size (to guarantee a fixed memory footprint for signal processors) and that array sizes are compile-time constants.

Definition 4.3 (Zero type). Each sample type T has an associated *zero* type, notated 0_{T}, defined by the rules below:

$$\begin{aligned} 0_{\mathsf{int}[l,h]} &= \mathsf{int}[0,0] \,; \\ 0_{\mathsf{float}[l,h]} &= \mathsf{float}[0.0,0.0] \,; \\ 0_{[n]\mathsf{T}} &= [n]0_{\mathsf{T}} \,. \end{aligned}$$

[7] Faust users can always introduce explicit min and/or max operations to explicitly bound intervals to make such functions total.

When V represents the set of sample values denoted by a type T, we know 0_{V} has[8] also type 0_{T}.

Definition 4.4 (Type operations). The type domain supports type operations that are either (1) extensions of usual arithmetic operations or (2) a least-upper bound operation $\sqcup$. For a generic type operator $\star$ (including $\sqcup$) and $N, N' \in \{\mathsf{int}, \mathsf{float}\}$, they are partially defined by the following rules:

$$\begin{aligned} \mathsf{int} \sqcup \mathsf{float} &= \mathsf{float} \,; \\ [l,h] \sqcup [l',h'] &= [\min(l,l'), \max(h,h')] \,; \\ N[l,h] \star N'[l',h'] &= (N \sqcup N')([l,h] \star [l',h']) \,; \\ [n]\mathsf{T} \star [n]\mathsf{T}' &= [n](\mathsf{T} \star \mathsf{T}') \,. \end{aligned}$$

under the sole conservative constraint that, for any arithmetic operator $\star$, $[l,h] \star [l',h'] = [L,H]$ such that, for all $x \in [l,h]$ and $x' \in [l',h']$, the value of $x \star x'$ is in $[L,H]$. Note that total arithmetic type operations are thus always definable, using the worst-case definition $[l,h] \star [l',h'] = [-\infty, +\infty]$.

4.2 Signal type environment Γ

Definition 4.5 (Signal-type environment). A signal type environment Γ is used to store type hypotheses for incoming signals. It maps each mutually recursive signal X_i and each input signal I_n to an appropriate signal type T^r:

$$\begin{aligned} \Gamma(X_i) &= N[l,h]^r, \quad N \in \{\mathsf{int}, \mathsf{float}\} \,; \\ \Gamma(\mathsf{I}_n) &= \mathsf{float}[-\infty, +\infty]^r \,, \end{aligned} \tag{3}$$

and, in each binding, for some appropriate N, l, h and r. As one can see, since Faust is a mostly audio-oriented DSL, it has been decided that only scalar signal types can be assigned to input signals. This means that communications with the outside world are limited to scalar signals (this scalar property is also checked for the resulting output signals), and that vectors can only be communicated in serialized form.

While the full audio range is often represented by the interval $[-1.0, +1.0]$ in practice, this restriction is not enforced in Faust and signals can use the full floating-point range.

Mutually recursive signals can be seen as additional scalar inputs and outputs, where each output is connected to the corresponding input via a 1-sample delay. The type and rate of the corresponding input and output must be the same. Even though our algorithm could probably handle arbitrary types on recursive signals, we indeed believe that taking a simpler approach is granted in the type of audio applications we envision.

4.3 Typing rules

The signal typing rules $\Gamma \vdash (E, D) : \mathsf{T}^r$ are provided below by induction on E. Since D is seldom used, we note E the pair (E, D) when D is not used locally in a given rule.

Definition 4.6 (Constant signals typing rules).

$$\frac{}{\Gamma \vdash k : \mathsf{int}[k,k]^1} \tag{Int}$$

$$\frac{}{\Gamma \vdash f : \mathsf{float}[f,f]^1} \tag{Float}$$

[8] We say that value v has type T, noted $v : \mathsf{T}$, when $v \in \mathbb{M}(\mathsf{T})$.

Definition 4.7 (Input and recursive signals).

$$\frac{}{\Gamma \vdash \mathsf{I}_n : \Gamma(\mathsf{I}_n)} \quad \text{(Input)}$$

$$\frac{\Gamma \vdash (E_i, D) : \Gamma(X_i) \quad D(X) = \langle E_0 E_1 \dots E_{n-1} \rangle \quad i \in [0, n-1]}{\Gamma \vdash (X_i, D) : \Gamma(X_i)} \quad \text{(Recursive)}$$

The typing rule for recursive signals is unusual in that, in addition to specifying that the type of a recursive signal identifier is directly available in the typing environment Γ, it also checks that the corresponding recursive definition present in D is properly typed.

Definition 4.8 (Numerical operations).

$$\frac{\Gamma \vdash E_i : \mathsf{T}_i^r \quad \mathsf{T} = \mathsf{T}_1 \star \mathsf{T}_2}{\Gamma \vdash E_1 \star E_2 : \mathsf{T}^r} \quad \text{(Op)}$$

Definition 4.9 (Delay operation).

$$\frac{\Gamma \vdash E_1 : \mathsf{T}_1^r \quad \Gamma \vdash E_2 : \mathsf{int}[k_1, k_2]^r \quad 0 \le k_1 \le k_2}{\Gamma \vdash E_1 @ E_2 : (\mathsf{T}_1 \sqcup 0_{\mathsf{T}_1})^r} \quad \text{(Delay)}$$

Definition 4.10 (Up and Down sampling).

$$\frac{\Gamma \vdash E : \mathsf{T}^r}{\Gamma \vdash E \uparrow^n : \mathsf{T}^{nr}} \quad \text{(Up)}$$

$$\frac{\Gamma \vdash E : \mathsf{T}^{nr}}{\Gamma \vdash E \downarrow^n : \mathsf{T}^r} \quad \text{(Down)}$$

Definition 4.11 (Vectorize and Serialize).

$$\frac{\Gamma \vdash E : \mathsf{T}^{nr}}{\Gamma \vdash \mathsf{v}(E, n) : [n]\mathsf{T}^r} \quad \text{(Vectorize)}$$

$$\frac{\Gamma \vdash E : [n]\mathsf{T}^r}{\Gamma \vdash \mathsf{s}(E) : \mathsf{T}^{nr}} \quad \text{(Serialize)}$$

$$\frac{\Gamma \vdash E_i : [n_i]\mathsf{T}_i^r \quad \mathsf{T} = \mathsf{T}_1 \sqcup \mathsf{T}_2}{\Gamma \vdash E_1 \# E_2 : [n_1 + n_2]\mathsf{T}^r} \quad \text{(Concat)}$$

$$\frac{\Gamma \vdash E_1 : [n_1]\mathsf{T}_1^r \quad \Gamma \vdash E_2 : \mathsf{int}[k_1, k_2]^r \quad 0 \le k_1 \quad k_2 < n_1}{\Gamma \vdash E_1[E_2] : \mathsf{T}_1^r} \quad \text{(Access)}$$

Definition 4.12 (Type/rate-correct signal). One says that a signal expression (E, D) is *type/rate-correct* iff there exist Γ, T and r such that $\Gamma \vdash (E, D) : \mathsf{T}^r$.

Definition 4.13 (Signal type). A signal s is said to *have type* T^r, noted $s : \mathsf{T}^r$, iff $\mathrm{Dom}(s) = \mathbb{T}_r$ and, for all $t \in \mathbb{T}_r$, one has $s(t) : \mathsf{T}$.

Proposition 4.14 (Subject Reduction). *Let (E, D) be a causal and type/rate-correct signal expression, such that $\Gamma \vdash (E, D) : \mathsf{T}^r$. Then, if $a(\mathsf{I}) : \Gamma(\mathsf{I})$ for all $\mathsf{I} \in \mathrm{Dom}(\Gamma)$, then $\mathbb{S}_a^r(E) : \mathsf{T}^r$.*

Note that this proposition is not as obvious as might be thought at first: if the proposition $\mathrm{Dom}(\mathbb{S}_a^r(E)) \subseteq \mathbb{T}_r$ is obviously true (see Section 3), the fact that $\mathbb{S}_a^r(E)$ is defined on the whole T^r is a consequence of the typing constraints.

4.4 Sample-type inference

In all signal typing rules above but one, sample types and rates are independent. The only one that creates a dependence is (Serialize); the size of the vector of the incoming signal is needed to compute the rate of the output signal. Therefore, we can infer first the sample-types independently, and then the rates, once the type information is available. In this section, we will rewrite the typing rules by separating the sample-types and the rates, in a two-step process.

First, a new, simpler type environment, noted $\Omega = \Omega(\Gamma)$, is derived from Γ; it is just a mapping that associates to each x in the domain of Γ the value type T, discarding the rate information r present in $\Gamma(x) = \mathsf{T}^r$.

Second, the sample-type rules are defined as just the signal typing rules of Section 4 where rate information is erased. In each rule, all typing judgments $\Gamma \vdash (E, D) : \mathsf{T}^r$ are replaced by $\Omega \vdash (E, D) : \mathsf{T}$.

Lemma 4.15 (Sample Type Consistency). *If one has $\Gamma \vdash (E, D) : \mathsf{T}^r$, then $\Omega(\Gamma) \vdash (E, D) : \mathsf{T}$.*

Proof. Trivial. Since the datatypes of signal values do not depend on rates, removing rate information does not prevent data-only type checking to proceed. □

With this two-step approach, the role of the straightforward sample-type inference algorithm (not presented here) will just be to compute a sample type environment Ω for any given signal (E, D).

5 RATE INFERENCE

Starting with a tuple of expressions $L = \langle E_0, \dots, E_{n-1} \rangle$ that share a common recursive environment D and which, typically, represent the output signals of a Faust program, the rate inference algorithm is a three-stage process that

- extends L with all the recursive subexpressions that can be reached,
- infers a rate environment Δ_i for each expression E_i of the extended L,
- and forms Δ by combining the Δ_i together.

5.1 Rate environments

Definition 5.1 (Rate environment). A rate environment Δ is a mapping that associates to some input signal I_n and mutually recursive signal X_i a rate r.

Definition 5.2 (Joinable environments). Two environments Δ_1 and Δ_2 can be joined, written $\Delta_1 \simeq \Delta_2$, iff

$$\forall x \in \mathrm{Dom}(\Delta_1) \cap \mathrm{Dom}(\Delta_2),\ \Delta_1(x) = \Delta_2(x)\ . \quad (4)$$

Definition 5.3 (Union of joinable environments). The union of two joinable environments Δ_1 and Δ_2 is written $\Delta_1 \cup \Delta_2$. The resulting environment is such that:

$$\mathrm{Dom}(\Delta_1 \cup \Delta_2) = \mathrm{Dom}(\Delta_1) \cup \mathrm{Dom}(\Delta_2)\ ;$$
$$(\Delta_1 \cup \Delta_2)(x) = \begin{cases} \Delta_1(x), & \text{if } x \in \mathrm{Dom}(\Delta_1) \\ \Delta_2(x), & \text{if } x \in \mathrm{Dom}(\Delta_2)\ . \end{cases} \quad (5)$$

Definition 5.4 (Scaled environment). We note $n\Delta$ the environment Δ scaled by an integer factor n, such that:

$$\mathrm{Dom}(\Delta) = \mathrm{Dom}(n\Delta)\ ;$$
$$\forall x \in \mathrm{Dom}(\Delta),\ (n\Delta)(x) = n\Delta(x)\ . \quad (6)$$

Definition 5.5 (Rate-scalable expressions). A typable expression E is rate-scalable, written $\mathrm{Adj}(E)$, if one can scale, or "adjust", its rate r by scaling its environment, i.e., for any signal type environment Γ and type T:

$$\Gamma \vdash E : \mathsf{T}^r \Rightarrow n\Gamma \vdash E : \mathsf{T}^{nr} , \tag{7}$$

where the notion of rate environment scaling is straightforwardly extended to signal type environments Γ. The idea behind rate-scalable expressions is that some signals can have their rate adjusted, i.e., properly scaled, when the context in which they are used requires it, and others don't. For instance, the rate of $X_1 \downarrow^3 +2$ cannot be modified (constants such as 2 have a fixed rate, of 1), while the rate of $X_1 \uparrow^2 +I_0$, which only depends on the ones of X_1 and I_0, can indeed be modified if these do have to change. Our algorithm will take advantage of such flexibility whenever possible.

Proposition 5.6. *Expressions $E \downarrow^n$, $E \uparrow^n$, $\mathsf{v}(E, n)$ and $\mathsf{s}(E)$ are also rate-scalable, as are binary expressions, when, recursively, E is rate-scalable.*

Definition 5.7 (Uniform rate environment). A rate environment Δ is said *uniform* iff all x in $\mathrm{Dom}(\Delta)$ have the same rate scalability in Δ.

Environments of identifiers that are all rate-scalable are decorated with a rate scalability of 1: Δ^1. For other expressions, the environment will be decorated with a 0 value: Δ^0. In the sequel, all rate-scalable rate environments are uniform.

5.2 Combining rate environments

Definition 5.8 (Independent environments). Two rate environments Δ_1 and Δ_2 are independent iff

$$\mathrm{Dom}(\Delta_1) \cap \mathrm{Dom}(\Delta_2) = \emptyset . \tag{8}$$

Definition 5.9 (Dependent environments). Two rate environments Δ_1 and Δ_2 that are not independent are called dependent and thus satisfy the following equation:

$$\mathrm{Dom}(\Delta_1) \cap \mathrm{Dom}(\Delta_2) \neq \emptyset . \tag{9}$$

Definition 5.10 (Environment addition). The addition $\Delta_1^{\upsilon_1} + \Delta_2^{\upsilon_2}$ of two dependent environments $\Delta_1^{\upsilon_1}$ and $\Delta_2^{\upsilon_2}$ is (partially) defined as follows, for some $x \in \mathrm{Dom}(\Delta_1) \cap \mathrm{Dom}(\Delta_2)$:

$$\frac{r_i = \Delta_i(x) \quad m = \mathrm{lcm}(r_1, r_2) \quad (\frac{m}{r1})^{\upsilon_1}\Delta_1 \simeq (\frac{m}{r2})^{\upsilon_2}\Delta_2}{\Delta_1^{\upsilon_1} + \Delta_2^{\upsilon_2} \rightarrow ((\frac{m}{r1})^{\upsilon_1}\Delta_1 \cup (\frac{m}{r2})^{\upsilon_2}\Delta_2)^{\upsilon_1\upsilon_2}} \tag{10}$$

Basically, if there is an identifier x in the intersection of the Δ_i, we need to be sure they provide the same rate to x, to allow the Δ_i to be combined together. If one of these environments is fixed ($\upsilon_i = 0$), we just have to recover the rate associated to x there, namely $\Delta_i(x)$. However, if one environment (or both) is rate-scalable, we have more leeway, and can scale them to ensure they provide the same rate; we pick here the $\mathrm{lcm}(r_1, r_2)$ to be this common rate.

Two comments are warranted about this important rule. First, note that the above definition is well-formed; the exact choice of x in the rules above has no impact on the end result, since all rate environments are uniform, and so rates will be modified always in the same way. Second, even though the least common multiplier operation is usually applied to natural numbers, we use here its extension to rationals (see, e.g., [22]), where $\mathrm{lcm}(n_1/p_1, n_2/p_2) = \mathrm{lcm}(n_1, n_2)/\mathrm{gcd}(p_1, p_2)$. The important aspect here is that the ratio of such a least common multiplier with its n_i/p_i arguments is always a natural number.

Lemma 5.11. *The addition of two rate environments $\Delta_1^{\upsilon_1} + \Delta_2^{\upsilon_2}$ defined by the rule above is a (1) commutative and (2) associative operation.*

Proof.

(1) Commutativity is obvious, since $\simeq$ is an equivalence relation
(2) Associativity is a direct consequence of the associativity of the operators used to define rate environment addition: multiplication on rates, product of rate-scalabilities, set intersection and union, and lcm operation on rationals. □.

The domain of dependent rate environments is thus a commutative semigroup for addition.

5.3 Local rate inference rules

The rate inference algorithm described uses a set of rules

$$E, \Omega \rightarrow \langle \Delta^{\upsilon}, r \rangle$$

that express how to compute, for an expression E and sample-type environment Ω, a compatible rate environment Δ, with its rate-scalability υ, and a so-called *local* rate r. They are called local since they don't handle recursive signals, which is addressed in a subsequent phase. As usual, Ω is omitted when not needed locally.

Note that our algorithm infers, from rates for constants, inputs and recursive signals that are assumed to be integers, rates that are also integer, and not rational, as introduced above. Also, to show the flexibility of our approach, we fixed the rate scalability of constants to 0 and the ones of identifiers to 1; this could easily be parameterized by introducing a rate scalability environment, for instance to include fixed-rate input signals.

Definition 5.12 (Numbers). All numbers are of fixed rate 1. We note $\perp$ the rate environment with an empty domain.

$$\frac{}{k \rightarrow \langle \perp^0, 1 \rangle} \tag{Int}$$

$$\frac{}{f \rightarrow \langle \perp^0, 1 \rangle} \tag{Float}$$

Definition 5.13 (Inputs). All inputs, actual inputs but also inputs of recursive signals, are of rate-scalable rate, initially 1.

$$\frac{}{\mathsf{I}_n \rightarrow \langle \perp[\mathsf{I}_n \rightarrow 1]^1, 1 \rangle} \tag{Input}$$

$$\frac{}{X_i \rightarrow \langle \perp[X_i \rightarrow 1]^1, 1 \rangle} \tag{Recursive}$$

Definition 5.14 (Up-sampling and serialize).

$$\frac{E \rightarrow \langle \Delta^{\upsilon}, r \rangle}{E \uparrow^n \rightarrow \langle \Delta^{\upsilon}, nr \rangle} \tag{Upsampling}$$

$$\frac{\Omega \vdash E : [n]T \quad E \rightarrow \langle \Delta^{\upsilon}, r \rangle}{\mathsf{s}(E) \rightarrow \langle \Delta^{\upsilon}, nr \rangle} \tag{Serialize}$$

Definition 5.15 (Down-sampling and vectorize). The "-var" rule variants below are compatible with their non-"-var" versions; consistency is thus ensured.

$$\frac{E \to \langle \Delta^{\upsilon}, nr \rangle}{E \downarrow^{n} \to \langle \Delta^{\upsilon}, r \rangle} \quad \text{(Down)}$$

$$\frac{E \to \langle \Delta^{1}, r \rangle \;\; m = \mathrm{lcm}(n, r)}{E \downarrow^{n} \to \langle (\frac{m}{r}\Delta)^{1}, m/n \rangle} \quad \text{(Down-var)}$$

$$\frac{E \to \langle \Delta^{\upsilon}, nr \rangle}{\mathsf{v}(E, n) \to \langle \Delta^{\upsilon}, r \rangle} \quad \text{(Vectorize)}$$

$$\frac{E \to \langle \Delta^{1}, r \rangle \;\; m = \mathrm{lcm}(n, r)}{\mathsf{v}(E, n) \to \langle (\frac{m}{r}\Delta)^{1}, m/n \rangle} \quad \text{(Vect-var)}$$

Definition 5.16 (Monorate binary expressions). To handle in a general way binary expressions that force their subexpressions to have the same rate, we introduce, for the sake of simplicity, a generic monorate pairing operator between signal expressions, typed with a pair of types.

$$\frac{(E_1, E_2) \to \langle \Delta^{\upsilon}, r \rangle}{E_1 \star E_2 \to \langle \Delta^{\upsilon}, r \rangle} \quad \text{(Op)}$$

$$\frac{(E_1, E_2) \to \langle \Delta^{\upsilon}, r \rangle}{E_1 @ E_2 \to \langle \Delta^{\upsilon}, r \rangle} \quad \text{(Delay)}$$

$$\frac{(E_1, E_2) \to \langle \Delta^{\upsilon}, r \rangle}{E_1 \# E_2 \to \langle \Delta^{\upsilon}, r \rangle} \quad \text{(Concat)}$$

$$\frac{(E_1, E_2) \to \langle \Delta^{\upsilon}, r \rangle}{E_1[E_2] \to \langle \Delta^{\upsilon}, r \rangle} \quad \text{(Access)}$$

Definition 5.17 (Pairs of expressions).

$$\frac{E_i \to \langle \Delta_i^{\upsilon_i}, r_i \rangle \;\; \Delta_i' = \Delta_i[o \to r_i] \;\; \Delta'^{\upsilon_1}_1 + \Delta'^{\upsilon_2}_2 \to \Delta^{\upsilon}}{(E_1, E_2) \to \langle \Delta^{\upsilon}_{/o}, \Delta(o) \rangle} \quad \text{(Pair)}$$

where Δ_i' introduces o as a free identifier not in the domain of any of the Δ_i; here, one can think of o as a proxy for the result. Since it is used in the correctness proof, but serves no other purpose, it is removed in the end result environment ($\Delta_{/x}$ is identical to Δ, except it is undefined for x).

Proposition 5.18. *If $E, \Omega \to \langle \Delta^{1}, r \rangle$, then E is rate-scalable.*

5.4 Soundness

Definition 5.19 (Well-typed Mapping). A recursive mapping D is *well-typed* in Γ, written $\Gamma \vdash D$, iff, for all $X \in \mathrm{Dom}(D)$ and $i \in [0, \mathrm{length}(D(X)) - 1]$, one has $\Gamma \vdash (E_i, D) : \Gamma(X_i)$.

Definition 5.20. A pair (Ω, Δ) is *included* into Γ, written $(\Omega, \Delta) \sqsubset \Gamma$, iff, for all x in $\mathrm{Dom}(\Delta)$, one has $\Gamma(x) = \Omega(x)^{\Delta(x)}$.

Lemma 5.21. *If $(\Omega, \Delta_1 \cup \Delta_2) \sqsubset \Gamma$, then $(\Omega, \Delta_i) \sqsubset \Gamma$.*

Proof. By definition. □

Theorem 5.22 (Local Soundness). *Assume $\Omega \vdash (E, D) : \mathsf{T}$ and $(E, \Omega) \to \langle \Delta^{\upsilon}, r \rangle$, for some Δ, υ and r. Then, for any integer p and signal environment Γ, if one has*

- $(\Omega, p^{\upsilon}\Delta) \sqsubset \Gamma$
- *and* $\Gamma \vdash D$,

then $\Gamma \vdash (E, D) : \mathsf{T}^{p^{\upsilon} r}$.

This is the first of the main theorems of this paper. Assume that we have a signal expression E that includes recursive signals kept into the mapping D. Moreover, we assume that E, together with D, is properly sample-typed, i.e., it has a "standard" type T in some "standard" sample type environment Ω. Assume also that, when performing rate inference on (E, Ω) via the $\to$ relation, we derive some rate environment Δ and related rate scalability υ, together with a rate r for the signal E. Then, two cases can occur.

- Either $\upsilon = 0$, in which case $p^{\upsilon} = 1$ for all p, and thus we also assume that (Ω, Δ) is compatible with, i.e., is included into, some signal environment Γ (which includes rate information for identifiers), with which moreover D can be also properly typed.
- Or $\upsilon = 1$, in which case Δ is rate-scalable, and thus we can pick any p and Γ such that $(\Omega, p\Delta)$ is compatible with Γ, as long as D is properly typed.

Then, under all these assumptions, the theorem states that E with D has Type T and Rate r (or pr, if $\upsilon = 1$) in the type and rate environment Γ. The rate inference algorithm is thus sound.

Proof. By induction on E and case analysis.

Numbers. Trivial, with the (Int) and (Float) typing rules, since $\upsilon = 0$, $\Delta = \bot$ and $r = 1$.

Input. We have $\langle \Delta^{\upsilon}, r \rangle = \langle \bot[\mathsf{I}_n \to 1]^1, 1 \rangle$.

By definition of $\sqsubset$ with $\upsilon = 1$, we know $\Gamma = \Gamma_0[\mathsf{I}_n \to \mathrm{float}[-\infty, +\infty]^p]$ for some Γ_0. Since $\upsilon = 1$ and $r = 1$, we have $\mathsf{T}^{p^{\upsilon} r} = \Omega(\mathsf{I}_n)^p = \mathrm{float}[-\infty, +\infty]^p$. We thus get the conclusion by the (Input) typing rule.

Recursive. We have $\langle \Delta^{\upsilon}, r \rangle = \langle \bot[X_i \to 1]^1, 1 \rangle$.

By definition of $\sqsubset$ with $\upsilon = 1$, we know $\Gamma = \Gamma_0[X_i \to \Omega(X_i)^p]$ for some Γ_0. Since, moreover, $r = 1$, we have $\mathsf{T}^{p^{\upsilon} r} = \Omega(X_i)^p$. Also, since $\Gamma \vdash D$, we know that for all $X \in \mathrm{Dom}(D)$ and $X_i \in \mathrm{Dom}(X)$, one has $\Gamma \vdash E_i : \Gamma(X_i)$. Thus, by application of the (Recursive) typing rule, we deduce that $\Gamma \vdash X_i : \Gamma(X_i)$, yielding the conclusion.

Upsampling. We have $E = E' \uparrow^{n}$ and $\langle \Delta^{\upsilon}, r \rangle = \langle \Delta^{\upsilon}, nr' \rangle$, with $E' \to \langle \Delta^{\upsilon}, r' \rangle$.

By the (Up) sample typing rule, $\Omega \vdash E : \mathsf{T}$ implies that $\Omega \vdash E' : \mathsf{T}$. Thus, by induction, we get $\Gamma \vdash E' : \mathsf{T}^{p^{\upsilon} r'}$.

By the (Up) typing rule, we then get $\Gamma \vdash E : \mathsf{T}^{np^{\upsilon} r'}$. Since $r = nr'$, then, we obtain the conclusion.

Serialize. We have $E = \mathsf{s}(E')$ and $\langle \Delta^{\upsilon}, r \rangle = \langle \Delta^{\upsilon}, nr' \rangle$, with $E' \to \langle \Delta^{\upsilon}, r' \rangle$ with $\Omega \vdash E' : [n]\mathsf{T}'$.

By induction, we get $\Gamma \vdash E' : [n]\mathsf{T}'^{p^{\upsilon} r'}$.

By the (Serialize) typing rule, we then get $\Gamma \vdash E : \mathsf{T}'^{np^{\upsilon} r'}$. Since $r = nr'$, then, we conclude $\Gamma \vdash E : \mathsf{T}'^{p^{\upsilon} r}$.

Down. We have $E = E' \downarrow^{n}$ and $\langle \Delta^{\upsilon}, r \rangle$ such that $E' \to \langle \Delta^{\upsilon}, r' \rangle$ and $r' = nr$.

By the (Down) sample typing rule, $\Omega \vdash E : \mathsf{T}$ implies that $\Omega \vdash E' : \mathsf{T}$. Thus, by induction, $\Gamma \vdash E' : \mathsf{T}^{p^{\upsilon} r'}$.

By the (Down) typing rule, since $\mathsf{T}^{p^{\upsilon} r'} = \mathsf{T}^{p^{\upsilon} nr}$, we then get $\Gamma \vdash E : \mathsf{T}^{p^{\upsilon} r}$, yielding the conclusion.

Down-var. We have $E = E' \downarrow^{n}$ and $\langle \Delta^{\upsilon}, r \rangle = \langle (\frac{m}{r'}\Delta')^1, m/n \rangle$ such that $E' \to \langle \Delta'^1, r' \rangle$ and $m = \mathrm{lcm}(n, r')$. We also assume that $(\Omega, p\Delta) \sqsubset \Gamma$, since $\upsilon = 1$.

By the (Down) sample typing rule, $\Omega \vdash E : \mathsf{T}$ implies that $\Omega \vdash E' : \mathsf{T}$. Thus, by induction on E', with $p\frac{m}{r'}$ as p' and $v' = 1$ in this inductive step, we get $\Gamma \vdash E' : \mathsf{T}^{p\frac{m}{r'}r'}$, i.e., $\Gamma \vdash E' : \mathsf{T}^{pm}$.

By the (Down) typing rule, we then get $\Gamma \vdash E : \mathsf{T}^{pm/n}$, yielding the conclusion.

Vectorize and Vect-var. Same reasoning as for (Down) and (Down-var).

Binary expressions. Direct consequence of the proof for pairs of expressions.

Pair of expressions. We have $E = (E_1, E_2)$ and $E \to \langle \Delta^v_{/o}, r\rangle$ such that $E_i \to \langle \Delta_i^{v_i}, r_i\rangle$, $\Delta'^{v_1}_1 + \Delta'^{v_2}_2 \to \Delta^v$ and $r = \Delta(o)$. We proceed by case on the pair (v_1, v_2). Note that the removal of Identifier o from Δ has no influence on the induction steps taken in the proof below, since they are irrelevant in the typing of E_i.

$(0,0)$. By definition of the rate environment addition, we have $\Delta^v = (\Delta'_1 \cup \Delta'_2)^0$ with $\Delta'_1 \simeq \Delta'_2$. Thus, for all x in $\mathrm{Dom}(\Delta'_1) \cap \mathrm{Dom}(\Delta'_2)$ and thus o, one has $\Delta(x) = \Delta'_1(x) = \Delta'_2(x)$. This entails $r_1 = \Delta'_1(o) = \Delta'_2(o) = r_2$.
By induction, since we know that $(\Omega, \Delta'_1 \cup \Delta'_2) \sqsubset \Gamma$ implies $(\Omega, \Delta'_i) \sqsubset \Gamma$ and thus $(\Omega, \Delta_i) \sqsubset \Gamma$, we have $\Gamma \vdash E_i : \mathsf{T}_i^{r_i}$. Thus, with $r_1 = r_2 = \Delta(o)$ and $\mathsf{T} = (\mathsf{T}_1, \mathsf{T}_2)$, we get the conclusion $\Gamma \vdash E : \mathsf{T}^r$.

$(0,1)$. We have $\Delta^v = (\Delta'_1 \cup n\Delta'_2)^0$, for $n = m/r_2$.
Here, one can show, similarly as above with $\Delta'_1 \simeq n\Delta'_2$, that $r_1 = nr_2$.
By induction, since $(\Omega, (\Delta'_1 \cup n\Delta'_2)) \sqsubset \Gamma$ implies $(\Omega, n^{v_i}\Delta'_i) \sqsubset \Gamma$ and thus $(\Omega, n^{v_i}\Delta_i) \sqsubset \Gamma$, we have (using $p_2 = n$) that $\Gamma \vdash E_i : \mathsf{T}_i^{n_i r_i}$ with $n_i = n^{v_i}$. Thus, with $n^0 r_1 = n^1 r_2 = \Delta(o)$ and $\mathsf{T} = (\mathsf{T}_1, \mathsf{T}_2)$, we get the conclusion $\Gamma \vdash E : \mathsf{T}^r$.

$(1,0)$. This is the symmetrical case of the previous one.

$(1,1)$. Here, $r = \Delta(o) = \mathrm{lcm}(r_1, r_2)$. The rate addition rule ensures that $\Delta^v = (\frac{r}{r_1}\Delta'_1 \cup \frac{r}{r_2}\Delta'_2)^1$.
Since $(\Omega, p\Delta) \sqsubset \Gamma$ implies $(\Omega, p\frac{r}{r_i}\Delta'_i) \sqsubset \Gamma$ and thus $(\Omega, p\frac{r}{r_i}\Delta_i) \sqsubset \Gamma$, we have, by induction with $p_i = p\frac{r}{r_i}$, that $\Gamma \vdash E_i : \mathsf{T}_i^{p\frac{r}{r_i}r_i}$. Thus, with $p\frac{r}{r_i}r_i = pr = p\Delta(o)$ and $\mathsf{T} = (\mathsf{T}_1, \mathsf{T}_2)$, we get the conclusion $\Gamma \vdash E : \mathsf{T}^{pr}$. □

5.5 Integer Completeness

We define an integer-only version $\Gamma \vdash_{\mathbb{N}} E : \mathsf{T}^R$ of the $\Gamma \vdash E : \mathsf{T}^R$ typing relation. In $\vdash_{\mathbb{N}}$, it is assumed that all derivation trees use only integer rates.

Theorem 5.23 (Local Integer Completeness). *If* $\Gamma \vdash_{\mathbb{N}} E : \mathsf{T}^R$, *then there exist* $\Delta, v \in \{0,1\}, r \in \mathbb{N}$ *and* $k \in \mathbb{N}$ *such that* $(E, \Omega(\Gamma)) \to \langle \Delta^v, r\rangle$ *with* $R = rk^v$ *and* $(\Omega(\Gamma), k^v\Delta) \sqsubseteq \Gamma$.

This is the second important theorem of this paper. Assume that there exists a derivation, involving only integer rates, of the type and integer rate T^R of a signal expression E in a signal environment Γ. Then, this theorem states that the rate inference algorithm $(E, \Omega(\Gamma)) \to \langle \Delta^v, r\rangle$ will succeed in finding a rate environment Δ with a rate-scalability v and (integer) rate r. Moreover, the inferred rate r will be minimal in the sense that, if the expression is found rate-scalable ($v = 1$), then the derived rate R will be an integer multiple of r and the multiplication factor ($k = R/r$) is fine-tuned to make all signals in Δ compatible, after multiplication with k, with the signal environment Γ.

Proof. By induction on E and case analysis, assuming $\Gamma \vdash_{\mathbb{N}} E : \mathsf{T}^R$. As usual, $\Omega(\Gamma)$ is omitted when not needed.

Numbers. Trivial, with the (Int) and (Float) typing rules, with $\Delta = \bot, v = 0, r = 1$ and any k.

Input. We just have to choose Δ, v and r such that $\langle \Delta^v, r\rangle = \langle \bot[\mathsf{I}_n \to 1]^1, 1\rangle$. Since we know that $\Gamma \vdash_{\mathbb{N}} \mathsf{I}_n : \mathsf{T}^R$, finally just choose $k = R$ to ensure the conclusion.

Recursive. This case is the same as for inputs (note that the usually tricky handling of recursive types does not translate when only looking at rates).

Upsampling. We have $E = E' \uparrow^n$ and, using the (Up) typing rule, we have $\Gamma \vdash_{\mathbb{N}} E' : \mathsf{T}^{R'}$ with $R = nR'$.

By induction, there exist a tuple $\langle \Delta'^{v'}, r'\rangle$ and k' such that $(E', \Omega(\Gamma)) \to \langle \Delta'^{v'}, r'\rangle$ and $R' = r'k'^{v'}$.

Applying the (Up) sampling step of the rate inference algorithm, we just choose $\langle \Delta^v, r\rangle = \langle \Delta'^{v'}, nr'\rangle$. We are left with finding k such that $R = rk^v$, i.e., such that $nR' = nr'k^{v'}$ or $R' = r'k^{v'}$. We just choose $k = k'$.

Serialize. We have $E = \mathsf{v}(E', n)$. This case is similar to the upsampling case, except that, in addition, one needs to use the Sample Type Consistency property to establish $\Omega(\Gamma) \vdash E' : [n]T'$, which is needed to execute the (Serialize) algorithm step.

Down. We have $E = E' \downarrow^n$ and, using the (Down) typing rule, we have $\Gamma \vdash_{\mathbb{N}} E' : \mathsf{T}^{R'}$ with $nR = R'$.

By induction, there exist a tuple $\langle \Delta'^{v'}, r'\rangle$ and k' such that $(E', \Omega(\Gamma)) \to \langle \Delta'^{v'}, r'\rangle$ and $R' = r'k'^{v'}$.

There are three cases.

- First, assume there is r'' such that $r' = nr''$. Using the (Down) algorithm step, pick $\langle \Delta^v, r\rangle = \langle \Delta'^{v'}, r''\rangle$. Then, we only have to find k such $R = r''k^v$, i.e., $nR = nr''k^v = r'k^v$, i.e., such that $R' = r'k^{v'}$. Thus, just choose $k = k'$ to complete the step.
- Then, assume that there is no such r'' and $v' = 0$. This would make the rate inference algorithm fail. Yet, such a case is impossible, since, having both $R' = r'k^0$ by induction and $R' = nR$ by typing, the rate r' must be a multiple of n, a contradiction.
- Finally, we have $v' = 1$, which enables the (Down-var) algorithm step.
  Here, we choose $\langle \Delta^v, r\rangle = \langle \frac{m}{r'}\Delta'^1, m/n\rangle$, where $m = \mathrm{lcm}(n, r')$. We need to find k such $R = (m/n)k^v$, i.e., such that $nR = mk^1$. We choose $k = r'k'/m$,
  First, k is indeed an integer. We have $r'|r'k'$ and $n|r'k'$, since $r'k' = R' = nR$. Thus $m = \mathrm{lcm}(n, r')|r'k'$. Moreover, $k\Delta = r'k'/m(\frac{m}{r'}\Delta') = k'\Delta'$, and thus $(\Omega(\Gamma), k^v\Delta) \sqsubseteq \Gamma$, completing the step.

Vectorize. Use the same reasoning as for (Down).

Pair of expressions. We have $E = (E_1, E_2)$. Using the (Op) typing rule, we also have $\Gamma \vdash_{\mathbb{N}} E_i : \mathsf{T}_i{}^R$.

By induction, there exist tuples $\langle \Delta_i^{v_i}, r_i\rangle$ and k_i such that $(E_i, \Omega(\Gamma)) \to \langle \Delta_i^{v_i}, r_i\rangle$, with the conditions $R = r_i k_i^{v_i}$ and $(\Omega(\Gamma), k_i^{v_i}\Delta_i) \sqsubseteq \Gamma$.

Using the (Pair) algorithm step, choose $\Delta^v = \Delta'^v_{/o}$ with $\Delta'^v = (\Delta'^{v_1}_1 + \Delta'^{v_2}_2)^{v_1 v_2}$ and $r = \Delta'(o)$. This environment addition is well-defined, since (1) one can take $x = o$ in the antecedent of the definition of addition and (2) $(\frac{m}{r_1})^{v_1}\Delta'_1 \simeq (\frac{m}{r_2})^{v_2}\Delta'_2$, with $m = \mathrm{lcm}(r_1, r_2)$.

Property (2) is true by the Environment Scaling Equivalence lemma (see below) for Δ_1 and Δ_2, since, by induction, we have $(\Omega(\Gamma), k_i^{v_i}\Delta_i) \sqsubseteq \Gamma$.

It is, also, true for o, present in both Δ'_i. Indeed, $\Delta'_i(o) = (\frac{m}{r_i})^{v_i} r_i = (\frac{m}{r_i})^{v_i}(R/k_i^{v_i}) = (\frac{mk_i^{v_i}}{R})^{v_i}(R/k_i^{v_i})$, which is $m^{v_i}R^{1-v_i}$, since $v_i v_i = v_i$. Since $R = r_1 k_1^{v_1} = r_2 k_2^{v_2}$, one has $R = m'm$ for some m'. Thus, we get $\Delta'_i(o) = m^{v_i}(m'm)^{1-v_i} = mm'^{1-v_i}$, and two cases occur.

- When the values of v_i are equal, we obtain the sought equality.
- Assume then, wlog, that $v_1 = 0$ and $v_2 = 1$. We need to show that m (for $v_2 = 1$) is equal to R (for $v_1 = 0$). Indeed, in this particular case, $m = \mathrm{lcm}(r_1, r_2) = \mathrm{lcm}(R, R/k_2) = R$.

We can now proceed to proving the two conclusion conditions of the theorem.

- We need k such that $R = \Delta'(o)k^{v_1 v_2} = (\frac{m}{r_i})^{v_i} r_i k^{v_1 v_2}$. First, assume $v_1 = v_2$. Then, R must be equal to $(\frac{m}{r_1})^{v_1} r_1 k^{v_1 v_2}$. Choose $k = r_1 k_1^{v_1}/m$ (which is an integer since $r_1 | r_1 k_1^{v_1}$ and $r_2 | r_2 k_2^{v_2} = r_1 k_1^{v_1}$). Then R must be $(\frac{m}{r_1})^{v_1} r_1 (r_1 k_1^{v_1}/m)^{v_1 v_2}$, which is equal to $(\frac{m}{r_1})^{v_1} r_1 (r_1 k_1/m)^{v_1}$, since $v_1 v_2 = v_1$. After simplification, we get $R = r_1 k_1^{v_1}$, which is true.
  In the second case, the v_i are different. Then, wlog, we take $v_1 = 0$. After simplifying the formula for R, we need to show that $R = m$. This is indeed true, since $R = r_1 = r_2 k_2$ and k_2 is an integer, leading to $m = r_1 = R$.
- We need to show that $(\Omega(\Gamma), k^v\Delta) \sqsubseteq \Gamma$, with $v = v_1 v_2$.
  First, assume $v_1 = v_2$. We have set $k = r_1 k_1/m$, which gives $k^{v_1 v_2}\Delta = (r_1 k_1/m)^{v_1}(\frac{m}{r_1})^{v_1}\Delta_1$, which yields $k^{v_1 v_2}\Delta = k_1^{v_1}\Delta_1$, giving the conclusion.
  In the second case, with different v_i and $v_1 = 0$, we need to show $(\Omega(\Gamma), \Delta) \sqsubseteq \Gamma$. It is a direct consequence of the induction for Δ_1. For the Δ_2 part of Δ, we need to show that $(\Omega(\Gamma), (\frac{m}{r_2})^1\Delta_2) \sqsubseteq \Gamma$. Yet, we have $R = r_1 = r_2 k_2$, yielding $m = r_1$; we need $(\Omega(\Gamma), (\frac{r_1}{r_2})\Delta_2) \sqsubseteq \Gamma$, i.e., $(\Omega(\Gamma), k_2\Delta_2) \sqsubseteq \Gamma$, which is true, by induction. □

Lemma 5.24 (Environment Scaling Equivalence). *If $k_1^{v_1}\Delta_1 \simeq k_2^{v_2}\Delta_2$ and $r_1 k_1^{v_1} = r_2 k_2^{v_2}$, then one has $(\frac{m}{r_1})^{v_1}\Delta_1 \simeq (\frac{m}{r_2})^{v_2}\Delta_2$, with $m = \mathrm{lcm}(r_1, r_2)$.*

Proof. For all elements in $\Delta_1 \cap \Delta_2$, by case on v_i. □

5.6 Rate inference

Rate inference is performed by first computing the local rates of the expressions E_i in the list L of signal outputs; all recursively defined signals used in E_i are gathered in a mapping D. If a rate can be successfully inferred for every expression E_i, the next step is to compute a global Δ by combining all these $\Delta_i^{v_i}$ environments. At the end of this process, one compute a reduced rate environment Δ. The last point is then to check that all recursive signals have the same input and output rate. The rate inference algorithm is provided in Figure 5.

```
rates(L, D) :
%-- Input:
%--   List L of n signal outputs E_i
%--   Mapping D for recursive signals
%-- Output:
%--   Typing environment Γ
%--   (with Γ(o_i) the type/rate of E_i)

%-- Infer types and local rates
for each E_i in L
    (Ω_i, T_i) = sample_type((E_i, D));
    (Δ_i^{v_i}, r_i) = local_rate((E_i, Ω_i));

%-- Compute the global sample type
%-- environment
Ω = ⋃_{i=0}^{n-1} Ω_i[o_i → T_i];

%-- Compute the global rate environment
Rs = ⋃_{i=0}^{n-1} {Δ_i[o_i → r_i]^{v_i}};
while ∃ intersecting R_1 and R_2 in Rs
    Rs = Rs ∪ {R_1 + R_2} − {R_1, R_2};
Δ = ⋃_{R∈Rs} R;

%-- Build the global signal type
%-- environment
Γ = [];
for each x in Dom(Δ)
    Γ = Γ[x → Ω(x)^{Δ(x)}];

%-- Check recursive signals
for each X in Dom(D)
    for each i from 0 to length(D(X)) − 1
        T_i^{r_i} = type/rate(Γ, (π_i(D(X)), D));
        check (T_i^{r_i} == Γ(π_i(X)));
return Γ;
```

Figure 5: `rates` **signal rate inference algorithm**

Definition 5.25 (Type/Rate Correctness). A list L of n signals = $\langle E_0, , ..., E_{n-1}\rangle$, with its recursive mapping D, is *type/rate correct* iff

there exists Γ such that, for all E_i, there exist type T_i and rate r_i such that $\Gamma \vdash (E_i, D) : \mathsf{T}_i^{r_i}$.

Theorem 5.26 (Rate Inference Soundness). *If the call to the algorithm* `rates`(L, D) *returns* Γ*, then* L *is type/rate correct for* Γ.

Proof. By definition of type/rate correctness, and picking `rates`(L, D) for Γ, one needs to show that, for all E_i, there exist a type and rate. For this, for each expression E_i, we use the Local Rate Inference Soundness theorem, after checking that each of its conditions is valid.

(1) The first condition, $\Omega_i \vdash (E_i, D) : \mathsf{T}_i$, is satisfied, since sample type inference is performed on each E_i with the `sample_type` algorithm (not described here). Use then the weakening typing rule, from Ω_i to Ω.
(2) The second condition, $(E_i, \Omega_i) \to \langle \Delta_i^{v_i}, r_i \rangle$, is ensured by the calls to `local_rate`.
(3) For the third condition, using $p = \Delta(o_i)/r_i$, we see that Γ is built so that $(\Omega, p^{v_i}\Delta_i) \sqsubset \Gamma$.
(4) The fourth and final condition, $\Gamma \vdash D$, is satisfied by the final checks on Γ.

Thus, by Local Rate Inference Soundness, proper sample type $\Omega(o_i)$ and rate $\Delta(o_i)$ exist for each expression E_i. □

Theorem 5.27 (Minimum Rate). *The rates provided by* `rates` *are minimal.*

Proof. Consequence of the Local Rate Inference Integer Completeness theorem, since all k there are integers. □

6 RELATED WORK

As a specifically audio-oriented DSL, Faust takes its roots into at least three domains: functional languages, music languages and synchronous languages. The last two families are strongly related to clocking issues.

Music languages such as Faust or Csound [5] make a clear distinction between audio rates, the pervasive digital audio sample rate information (44 kHz or 48 kHz), and control rates (`kr` in Csound parlance), related to the frequency at which, for instance, user interface components are sampled. The distinction is, in fact, mostly motivated by performance issues. Our rate information provides a finer-grained and more flexible way to handle a wide variety of rate requirements.

Regarding synchronous languages, the most relevant references are the ones related to the Synchronous Dataflow Model (see Lee *et al*'s seminal work [15], or [3] for a more recent, parameterized variant). In fact, our work can be seen as both (1) a reframing of the solving of "balance equations" in SDF [14] in the framework of annotated type systems [16] and (2) an extension of this scheme to rational rates. Contrarily to Lee *et al*'s global, integer matrix-based version, our rate algorithm is defined by induction on the syntax of expressions, allowing for the early and precise detection of rate inconsistencies and type theoretic-like correctness proofs. Our technique can also handle explicit rate and type constraints on input-output signals; in particular, typical audio environments expect them to carry scalar values, with no buffering required and fixed rates. More generally, our type and rate scheme is intended to include more involved typing and rational rating conditions (see Section 7). We believe that our static semantics approach, complimentary to the one usually adopted in the literature, is thus flexible.

Moving to other synchronous languages such as Lustre [7], Signal [4] or Lucid Synchrone [6], to mention a few, Faust does not attempt to provide the wide spectrum of clocking and data manipulation specifications present in these general-purpose synchronous languages. The emphasis is, as presented in the introduction, to match audio DSP features and their associated hard real-time, efficiency requirements. If our rational model for rates can be seen, in some sense, as a special case of the more abstract clocking mechanisms provided in these frameworks, i.e., "clocks as abstract types" [9] or integer clocks [11], we believe that the tight intertwining of our rate model and efficient inference algorithm will provide value to Faust users.

A recent language design that uses ideas similar to the ones found in Faust is Sig [21]. It is based on a functional paradigm, targets domain-specific applications (including music), uses discrete clocks, provides both visual and textual interfaces and only computes total functions. Yet, its design is clearly oriented towards providing abstract concepts on top of a synchronous framework, e.g., symbolic computation and meta-programming features, while Faust is geared towards efficient implementation, as shown for instance by Faust's C-based backend compared to Sig's JVM reliance. Sig's multirate clock mechanism is also still an on-going work.

Even though some papers on type-based clock mechanisms mention explicit rate inference algorithms, most authors limit their covering of this topic to a few comments about Hindley-Milner-based schemes. Yet, interestingly, in [20], a more precise description of a clock inference system is provided. In some sense, our usage of scalability parameters can be seen as a way of handling the equivalent of "rate schemes" within the rate inference algorithm itself. But, in addition to the structural type information found in Hindley-Milner systems, our problem also addresses the algebra of rate annotations.

7 FUTURE WORK

The typing rules and inference of Sections 4 and 5 are probably too strict to be of real practical interest. In particular, constant signals, for instance numbers, have to be up-sampled to be used in any expression of rate $r > 1$, which is very inconvenient. Thus, this section includes possible future work related to relaxing the typing and rate rules, and mentions the probable impact on the rate inference process.

One possible evolution of the typing rules is to accept to combine signals of different rates provided one rate is a multiple of the other. This can be done with the introduction of a single *Rate coercion* rule, as follows, assuming $n \in \mathbb{N}^*$.

$$\frac{\Gamma \vdash E : \mathsf{T}^r}{\Gamma \vdash E : \mathsf{T}^{nr}} \qquad \text{(Rate coercion)}$$

All equations $r_1 = r_2$ that appeared in the strict rate inference algorithm, and had to be enforced via unification, have, in the relaxed rate inference algorithm, to be replaced by appropriate parametrized equalities of the form $n_1 r_1 = n_2 r_2$. Note that the introduction of the (Rate coercion) rule is equivalent to adding implicit up sampling operations in the language, thus allowing, in

theory, to get rid of the explicit (Upsampling) rule. We suggest to keep it nonetheless, if only for documentation purposes.

Another possibility is to use an explicit subrating rule:

$$\frac{\Gamma \vdash E_i : \mathsf{T}_i^{r_i} \quad \mathsf{T}^r = \mathsf{T}_1^{r_1} \star \mathsf{T}_2^{r_2}}{\Gamma \vdash E_1 \star E_2 : \mathsf{T}^r} \qquad \text{(Subrating)}$$

which would allow subrated expressions to be passed to primitive operations. We introduce here a natural extension of the $\star$ relation over sample types with

$$\mathsf{T}_1^{r_1} \star \mathsf{T}_2^{r_2} = (\mathsf{T}_1 \star \mathsf{T}_2)^{\max(r_1,r_2)}, \text{ if } \min(r_1,r_2) | \max(r_1,r_2) \ .$$

The difference between the two approaches is that, for instance, the constant signal 10 becomes a signal with multiple rates in the first case, while, with the second approach, it is the flexibility of the subrating rule that allows to pass 10 (with its rate of 1) to an operator where a signal of a different rate, 2 say, is expected, as would be the case in an expression such as $10 + 7\ \uparrow^2$. Referential transparency issues, and more experiments, can help decide which approach is best.

More unusual, and intriguing, would be to allow some sort of "contravariant" subrating on constant expressions. For instance, when connecting a constant signal K of value 10 at Rate 1 to a slow-going signal expression S (for instance a slider enabling some sort of user interfacing at Rate 1/100), it would interesting to allow K to be deemed equivalent rate-wise to this slower signal (note this is going the opposite way of the previous proposals, which would have forced S to go as fast as K, i.e., adopt a rate of 1). This makes sense, since our knowledge that K is constant makes it amenable to a slower rate without loss of information. Moreover, this information is explicitly present with our type system, K having the type $\mathrm{int}[10, 10]^1$ (and this behavior would be generalized automatically to all expressions proven constant by the type checker).

We are also envisioning the possibility of adding rate constraints explicitly, either at the language level or within the embedding sound architecture, for instance to enforce a particular I/O rate, required by the outside world (e.g., the fact that a particular rate must be an integer, say 44kHz). The best way of doing so is also a matter of more experimenting, at the language-design and usage levels.

Finally, we believe that the direction set out by this research work could be of possible use in other languages and paradigms that address timing issues. For instance, our semantic approach to discrete times could have possible applications in a computer music language such as Antescofo [10] or even Sig [21], while our rate inference system and algorithm could be linked to the notion of rate information encoded in "rate types" [2] or to Kronos multirate semantics [17].

8 CONCLUSION

We show in this paper how the Faust digital audio processing language, traditionally based on scalar monorate signals, can be extended to handle multidimensional multirate signals. Specifically, we provide a formal definition of a new Intermediate Representation for Faust extended to enable the handling of the multirate framework proposed in [12]. We show how such signals can be formally defined on a rational model of its clocking mechanism.

On the practical side, we designed a new (type and) multirate inference algorithm, for which both soundness and (relative) completeness theorems are specified and proven. A prototype implementation of this algorithm in the Faust compiler static semantics phase, in a experimental multirate version of Faust, is underway.

ACKNOWLEDGMENTS

Part of this project was funded by the ANR FEEVER project. We thank Emilio Gallego Arias and Olivier Hermant for their thorough proofreading of this paper, and the anonymous reviewers for their insightful suggestions.

REFERENCES

[1] K. Barkati and P. Jouvelot. Synchronous programming in audio processing: A lookup table oscillator case study. *ACM Comput. Surv.*, 46(2):24, 2013.
[2] Thomas W. Bartenstein and Yu David Liu. Rate types for stream programs. *SIGPLAN Not.*, 49(10):213–232, October 2014.
[3] V. Bebelis, P. Fradet, A. Girault, and B. Lavigueur. Bpdf: A statically analyzable dataflow model with integer and boolean parameters. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, pages 3:1–3:10, Piscataway, NJ, USA, 2013. IEEE Press.
[4] A. Benveniste, P. Le Guernic, and C. Jacquemot. Synchronous programming with events and relations: the Signal language and its semantics. *Science of Computer Programming*, 16(2):103 – 149, 1991.
[5] R.C. Boulanger. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming.* MIT Press, 2000.
[6] P. Caspi, Gr. Hamon, and M. Pouzet. Lucid synchrone: un langage pour la programmation des systèmes réactifs. In *Systèmes temps réel*. Lavoisier, 2007.
[7] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: A declarative language for real-time programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, pages 178–188, New York, NY, USA, 1987. ACM.
[8] Ranald Clouston, Aleš Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. *Programming and Reasoning with Guarded Recursion for Coinductive Types*, pages 407–421. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
[9] J.-L. Colaço and M. Pouzet. *Embedded Software: Third International Conference, EMSOFT 2003, Philadelphia, PA, USA, October 13-15, 2003. Proceedings*, chapter Clocks as First Class Abstract Types, pages 134–155. Springer Berlin, 2003.
[10] José Echeveste, Arshia Cont, Jean-Louis Giavitto, and Florent Jacquemard. Operational semantics of a domain specific language for real time musician-computer interaction. *Discrete Event Dynamic Systems*, 23(4):343–383, August 2013.
[11] A. Guatto. *A Synchronous Functional Language with Integer Clocks.* PhD thesis, Université de recherche PSL, France, 2016.
[12] P. Jouvelot and Y. Orlarey. Dependent vector types for data structuring in multirate Faust. *Comput. Lang. Syst. Struct.*, 37(3):113–131, July 2011.
[13] S. C. Kleene. General recursive functions of natural numbers. *Math. Annalen*, 112(1):727–742, 1936.
[14] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35, January 1987.
[15] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proc. of the IEEE*, 75(9):1235–1245, Sept 1987.
[16] F. Nielson. Annotated type and effect systems. *ACM Comput. Surv.*, 28(2):344–345, June 1996.
[17] Vesa Norilo. Kronos: A declarative metaprogramming language for digital signal processing. *Computer Music Journal*, 39(4):30–48, 2015.
[18] Y. Orlarey, D. Fober, and S. Letz. Syntactical and semantical aspects of Faust. *Soft Comput.*, 8(9):623–632, September 2004.
[19] J.O. Smith. *Introduction to Digital Filters: With Audio Applications.* Music signal proc. series. W3K, 2008.
[20] J. P. Talpin and S. K. Shukla. Automated clock inference for stream function-based system level specifications. In *Tenth IEEE International High-Level Design Validation and Test Workshop*, pages 63–70, Nov 2005.
[21] Baltasar Trancón Widemann and Markus Lepper. On-line synchronous total purely functional data-flow programming on the java virtual machine with sig. In *Proceedings of the Principles and Practices of Programming on The Java Platform*, PPPJ '15, pages 37–50, New York, NY, USA, 2015. ACM.
[22] R. Zazkis and J. Truman. From trigonometry to number theory... and back: Extending lcm to rational numbers. *Digital Experiences in Mathematics Education*, 1(1):79–86, 2015.