Hyperstream Processing Systems

Nonstandard Modeling of Continuous-Time Signals

Kohei Suenaga
Kyoto University
ksuenaga@kuis.kyoto-u.ac.jp

Hiroyoshi Sekine   Ichiro Hasuo
University of Tokyo
{nue_of_k,ichiro}@is.s.u-tokyo.ac.jp

Abstract

We exploit the apparent similarity between (discrete-time) *stream processing* and (continuous-time) *signal processing* and transfer a deductive verification framework from the former to the latter. Our development is based on rigorous semantics that relies on *nonstandard analysis (NSA)*.

Specifically, we start with a discrete framework consisting of a Lustre-like stream processing language, its Kahn-style fixed point semantics, and a program logic (in the form of a type system) for partial correctness guarantees. This stream framework is transferred *as it is* to one for *hyperstreams*—streams of streams, that typically arise from sampling (continuous-time) signals with progressively smaller intervals—via the logical infrastructure of NSA. Under a certain continuity assumption we identify hyperstreams with signals; our final outcome thus obtained is a deductive verification framework of signals. In it one verifies properties of signals using the (conventionally discrete) proof principles, like fixed point induction.

Categories and Subject Descriptors F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs—Logics of programs; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Denotational semantics

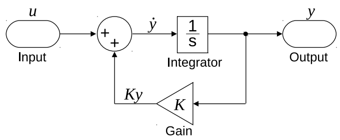
Keywords hybrid system; stream processing; signal processing; type system; nonstandard analysis

1. Introduction

Signal By *signals* we mean values that depend on continuous time, that is, functions $s : \mathbb{R}_{\geq 0} \to \mathbb{C}$.[1] Signals are everywhere in the real world: they are the most straightforward model of physical quantities like position, velocity, voltage, etc. Signals have been studied extensively in the theory of *dynamical systems*, or more recently from the engineering viewpoint of *control theory*.

[1] The use of complex numbers $\mathbb{C}$—instead of $\mathbb{R}$—as the range is due to our use of i (the imaginary unit) in our leading example in §6. This choice is not important: our theory behaves the same for both $\mathbb{C} \cong \mathbb{R}^2$ and $\mathbb{R}$.

POPL'13, January 23–25, 2013, Rome, Italy.



Hybrid system In the modern world where more and more physical systems are under the control of computers (cars, plants, pacemakers, etc.), signals that are not entirely smooth—with discrete *jumps* caused by digital control—have gained their significance. They play an important role in the study of *hybrid systems*, an aspect of the more general topic of *cyber-physical systems*. Simulink, an industry-standard tool for modeling and simulation of hybrid systems, supports the design of a hybrid system as a *signal processing system* composed of interconnected blocks (above right). A signal processing system is one that receives, processes and outputs signals.

Stream processing Study of formal verification—or computer science in general—has traditionally been focused on discrete data (this is changing rapidly, though). There *stream processing system* is a heavily studied notion, together with related notions like *dataflow network* and *reactive programming* [29]. A *stream* is an infinite sequence $(a_0, a_1, \dots)$ of data; thus it is a time-varying value $s : \mathbb{N} \to \mathbb{C}$ with a discrete notion of time. It bears an obvious similarity to the notion of signal. Moreover, common graphical presentations of stream processing systems look very much like Simulink block diagrams.

This similarity is the starting point of the current work. The difference between signals and streams is whether the time domain is continuous $(\mathbb{R}_{\geq 0})$ or discrete $(\mathbb{N})$. If one can unify this difference, the discrete techniques for streams that have been accumulated in computer science can be readily applied to signals. This is what we do, by the mathematical vehicle of *nonstandard analysis (NSA)*. NSA allows us to think of continuous-time *flow* dynamics as if it is a succession of discrete-time *jumps* each of which is infinitely small.

Nonstandard Analysis NSA is an alternative formalization of analysis—convergence, continuity, differentiation, etc.—that uses the explicit notion of *infinitesimal* (i.e. infinitely small) number. Leibniz's original formulation of analysis was based on infinitesimals; but a naive use of such immediately leads to a contradiction. It was Robinson [24] who gave a logically rigorous foundation for infinitesimals using the notion of *ultrafilter*.

In our previous work [13, 30] we used NSA to represent flow dynamics by means of `while` loops—each iteration of a loop changes values infinitesimally. What is remarkable about NSA is its logical infrastructure: its famous result called the *transfer principle* states that a formula is valid for real numbers if and only if it is valid for hyperreals (i.e. reals extended with infinitesimals). Therefore, reals vs. hyperreals (i.e. discrete vs. continuous, in the setting of [13, 30]) are *the same* from a logical point of view. This allowed us in [13, 30] to transfer a logical (i.e. deductive) verification technique for discrete programs *as it is* to hybrid systems.

In the current work we use the same idea to fill in the gap between signal processing and stream processing. We similarly transfer a deductive verification method, too. In it we employ the formalism of a (first-order functional) stream processing language SProc—it is modeled after the widely-accepted language Lustre [9]—to represent stream processing systems.

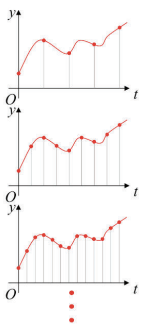
NSA is in fact not only about “analysis”: its use is found in many branches of mathematics, such as general topology [14, Chap. III] and posets [35]. In this paper we present another instance of such, namely *domain theory* upgraded with NSA (Appendix B, part of which already appeared in [3]). We use it in the definition of denotational semantics of $\mathrm{SProc}^{\mathtt{dt}}$.

Overview of the technical development On the right, in (1), is the overview of our development. It is centered around two key ideas: *hyperstream sampling* and *sectionwise execution*.

$$\begin{array}{c}
\boxed{\text{(Conti.) signal} \models^{?} \text{safety}} \\
\Uparrow \text{hyperstream sampling} \\
\boxed{\text{hyperstream} \models^{?} \text{safety}} \\
\Uparrow \text{Deductive verification} \\
\boxed{\mathrm{SProc}^{\mathtt{dt}} + \text{type system}} \\
\uparrow \text{Sectionwise execution} \\
\boxed{\mathrm{SProc} + \text{type system}}
\end{array} \qquad (1)$$

Hyperstream sampling Typically, a computer science approach to the study of continuous-time signals starts off with *sampling* a signal f. A sampling interval $\delta > 0$ results in a stream $(f(i\delta))_{i\in\mathbb{N}} = \big(f(0), f(\delta), f(2\delta), \dots\big)$. Obviously such sampling cannot be *exact*—we cannot know what happens to f during δ seconds of the sampling interval.

Then a natural idea is to consider infinitely many sampling intervals that are progressively small, as shown on the right. This results in a stream of streams $\Big(\big(f(\frac{i}{j+1})\big)_{i\in\mathbb{N}}\Big)_{j\in\mathbb{N}}$, that is



$$\left(\begin{array}{l}
\big(f(0), f(1), f(2), \dots\big), \\
\big(f(0), f(\frac{1}{2}), f(1), \dots\big), \\
\big(f(0), f(\frac{1}{3}), f(\frac{2}{3}), \dots\big), \\
\dots
\end{array}\right) . \qquad (2)$$

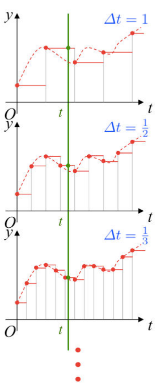
Roughly a *hyperstream* is such a stream of streams.

This *hyperstream sampling* still cannot be exact for all signals f: after all, there are only countably many sampling points, while $\mathbb{R}_{\geq 0}$ is uncountable. However, if f satisfies some continuity assumption, we could reconstruct the value $f(t)$ for $t \in \mathbb{R}_{\geq 0}$ as a certain limit of sampled values. Specifically, based on the figure below on the right, we “vertically” collect the following values from the sampling result (2).

$$f(\lceil t \rceil),\ f(\tfrac{\lceil 2t \rceil}{2}),\ f(\tfrac{\lceil 3t \rceil}{3}),\ \dots \quad \longrightarrow \quad f(t) \qquad (3)$$

This construction is called *smoothing*; here $\lceil \_ \rceil$ is “rounding up.”

We will formalize these sampling and smoothing operations (Smp and Smth) in §5. There we propose a class of functions (i.e. a continuity requirement) that makes hyperstream sampling indeed exact, that is, $\mathsf{Smth} \circ \mathsf{Smp} = \mathrm{id}$.



In this paper in fact we use a refined notion of hyperstream: it is not simply a stream of streams (described above) but is the $*$-*transform* of the notion of stream. The intuition behind this refined notion is: a hyperstream is a stream $(f(0), f(\mathtt{dt}), f(2\mathtt{dt}), \dots)$ with an infinitesimal sampling interval $\mathtt{dt}$. The j-th stream $\big(f(0), f(\frac{1}{j+1}), f(\frac{2}{j+1}), \dots\big)$ in (2) then occurs as its j-th approximation. $*$-transform is an NSA construction; its benefit is that we can transfer a logical theory of streams *as it is* to that of hyperstreams, via the celebrated *transfer principle* in NSA.

This idea of using an infinitesimal sampling interval is already presented in [3, §2.3], where they establish $\mathsf{Smth} \circ \mathsf{Smp} = \mathrm{id}$ for functions $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ that are everywhere *continuous* (they also hint an extension to piecewise continuity). Since we aim at hybrid applications, our class of functions (Def. 5.1) is broader and contains some Zeno examples such as a bouncing ball.

Sectionwise execution The second key idea is about integrating NSA into program semantics and transferring the latter from discrete to continuous/hybrid. For this purpose we use the idea called *sectionwise execution*; it was first used in our previous work [13, 30] for a language with `while` loops. Here we briefly review the idea as presented in [13, 30], adapting it later to the current setting of stream processing.

Its very first example is the program c_{elapse} on the right. Here `dt` is a constant that denotes an infinitesimal value; if that is the case the `while` loop will not terminate within finitely many steps. Nevertheless it is somehow intuitive to imagine the “execution” of the program to increase t from 0 to 1, in a smooth and continuous manner.

$$\begin{array}{l} t := 0\,; \\ \mathtt{while}\ t \leq 1\ \mathtt{do} \\ \quad t := t + \mathtt{dt} \end{array}$$

To put this intuition into rigorous program semantics, we think of the following *sectionwise execution*. For each natural number i we consider the i-*th section* of the program c_{elapse}, that is denoted by $c_{\mathsf{elapse}}|_i$ and shown on the right. Concretely, $c_{\mathsf{elapse}}|_i$ is obtained by replacing the infinitesimal `dt` in c_{elapse} with $\frac{1}{i+1}$. Informally $c_{\mathsf{elapse}}|_i$ is the “i-th approximation” of the original c_{elapse}.

$$\begin{array}{l} t := 0\,; \\ \mathtt{while}\ t \leq 1\ \mathtt{do} \\ \quad t := t + \frac{1}{i+1} \end{array}$$

A section $c_{\mathsf{elapse}}|_i$ does terminate within finite steps and yields $1 + \frac{1}{i+1}$ as the value of t. Now we collect the outcome of sectionwise execution and obtain a sequence

$$\big(\,1+1,\ 1+\tfrac{1}{2},\ 1+\tfrac{1}{3},\ \dots,\ 1+\tfrac{1}{i},\ \dots\big) \qquad (4)$$

which is intuitively thought of as a progressive approximation of the actual outcome of the original program c_{elapse}. Indeed, in the language of NSA, the sequence (4) represents a hyperreal number r that is infinitesimally close to 1.

In [30], based on this idea, we presented a framework for modeling and verification of hybrid systems. It consists of an imperative language While and a Hoare-style program logic Hoare, augmented with a constant `dt` and called $\mathrm{While}^{\mathtt{dt}}$ and $\mathrm{Hoare}^{\mathtt{dt}}$. Exploiting the transfer principle in NSA—which roughly states that reals and hyperreals are “logically the same”—we showed that the rules of $\mathrm{Hoare}^{\mathtt{dt}}$ (precisely the same as those of Hoare) are sound and relatively complete. Underlying is the denotational semantics of programs defined in the above sectionwise way. In [13] we applied several static analysis techniques (mainly for invariant discovery) to this setting. We also implemented an automated verification tool.[2]

It was speculated in [13, 30] that the use of `dt` is not only for `while` programs but is probably a general methodology for transferring a discrete verification framework to continuous/hybrid. Our current work is one such example: the framework of a stream processing language and a program logic (in the form of a type system) is transferred to the one for *hyperstream processing*. Here “sectionwise execution” takes the following concrete form. As a hyperstream processing language we introduce $\mathrm{SProc}^{\mathtt{dt}}$; it is a stream processing language SProc, augmented with a constant `dt` for an infinitesimal interval. The denotation $[\![p]\!]$ of an $\mathrm{SProc}^{\mathtt{dt}}$

[2] We note that “programs” in $\mathrm{While}^{\mathtt{dt}}$ are not executable in general; this is already clear in the example of c_{elapse}. We rather think of $\mathrm{While}^{\mathtt{dt}}$ as a *modeling* language, on which we can carry out static, deductive verification. The same is true of the language $\mathrm{SProc}^{\mathtt{dt}}$ introduced in the current paper.

program p—say p takes a hyperstream and returns a hyperstream—is defined by

$$[\![p]\!]\Bigl(\bigl((a_{i,j})_i\bigr)_j\Bigr) \;:=\; \Bigl([\![p|_j]\!]\bigl((a_{i,j})_i\bigr)\Bigr)_j \;; \tag{5}$$

that is, the section $p|_j$ (an SPROC program) is applied to the j-th stream of the input hyperstream, and their outcome is bundled up.

Example 1.1 (The sine curve)**.** Here is a program $\mathsf{pg}_{\mathsf{Sine}}$ in SPROC$^{\mathtt{dt}}$.

$$\begin{array}{l} \mathtt{node\ Sine()\ returns}\ (s) \\ \mathtt{where}\ s = 0\ \mathtt{fby}^1\ (s + c \times \mathtt{dt});\quad c = 1\ \mathtt{fby}^1\ (c - s \times \mathtt{dt}) \\ \mathtt{node\ Main()\ returns}\ (\mathtt{proj}_1\ \mathtt{Sine}()) \end{array} \tag{6}$$

The third line (declaring Main) is bureaucracy and can be ignored for the moment. The core part is the mutual recursive definition of the hyperstreams s and c, whose intuition we now explain. The operator $\mathtt{fby}^1$ means delay by one step (i.e. dt seconds):

$$(a_0, a_1, \dots)\ \mathtt{fby}^1\ (b_0, b_1, \dots) \;=\; (a_0, b_0, b_1, \dots) \;.$$

Therefore the (recursive) equation $s = 0\ \mathtt{fby}^1\ (s + c \times \mathtt{dt})$ means, for each n (that in fact ranges over *hypernatural numbers*),

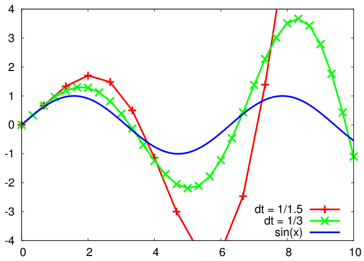
$$s(n) = \begin{cases} 0 & \text{if } n = 0, \\ s(n-1) + c(n-1) \times \mathtt{dt} & \text{otherwise.} \end{cases}$$

This yields the following equations.

$$s(0) = 0 \qquad \frac{s(n) - s(n-1)}{\mathtt{dt}} \;=\; c(n-1) \tag{7}$$

The value $s(n-1)$ is that of one step before $s(n)$, i.e. dt seconds before $s(n)$. Thus the equations (7) are identified with the differential equation $\sin'(t) = \cos(t)$ with the initial value $\sin(0) = 0$.

Intuitively, the section-wise execution of $\mathsf{pg}_{\mathsf{Sine}}$ realizes the sine curve as the limit of the approximations with $\mathtt{dt} = 1, \frac{1}{2}, \frac{1}{3}, \dots$. This is like the graphs on the right.



In the current work we employ a more advanced part of NSA than used in [13, 30]. It allows us to transfer statements not only on arithmetic facts but also on set-theoretical ones. More precisely, we can now transfer formulas of the first-order language $\mathscr{L}_X$ (Def. 2.5) that has $\in$ as a binary predicate. The details of this part of NSA is rather complicated but most of them are not needed; in §2.2 we list the minimal set of necessarily definitions and results.

A usage scenario Our technical framework summarized in (1) is supposed to be used in the following way.

We are given two data: a continuous-time signal f and a safety property P. The goal is to verify that f satisfies P. Towards that goal, one first *models* f by an SPROC$^{\mathtt{dt}}$ program pg_f. Typically we are not given f as a mathematical entity (i.e. a function $f : \mathbb{R}_{\geq 0} \to \mathbb{C}$), but we get its formal specification written in some formalism like ODEs and Simulink diagrams. In this case, modeling of f amounts to *translation* of a specification (say in ODEs) into an SPROC$^{\mathtt{dt}}$ program. This modeling part is briefly discussed in §5.3 but a more extensive treatment is left as future work.

The question then becomes whether the SPROC$^{\mathtt{dt}}$ program pg_f satisfies the safety property P. For that we can use a type system for SPROC$^{\mathtt{dt}}$: P is translated into a suitable type ν_{P}; and we try to derive a judgment $\mathsf{pg}_f \vdash \nu_{\mathsf{P}}$ using the typing rules. The typing rules include the well-established proof principle of *fixed point induction*.

Once the derivation is done, by type soundness (Thm. 4.20) it guarantees that the hyperstream $[\![\mathsf{pg}_f]\!]$ denoted by pg_f satisfies ν_{P}. Finally by Thm. 5.12 it implies that the signal $\mathsf{Smth}([\![\mathsf{pg}_f]\!])$ satisfies P. Thus we are done, under the condition that the SPROC$^{\mathtt{dt}}$ modeling of f is correct (i.e. $\mathsf{Smth}([\![\mathsf{pg}_f]\!]) = f$).

Note that the last paragraph is all about the metatheory. The actual verification task is derivation of a type judgment, and this is done in the same deductive style as the verification of (discrete-time) stream processing. In its course the NSA metatheory is completely concealed.

Organization of the paper In §2 we list the definitions and results of NSA that are used later. A prototype stream language SPROC is introduced in §3, together with its Kahn-style denotational semantics and a type system for safety guarantee. It is modeled after Lustre and is nothing novel; but the SPROC framework is carefully designed so that it allows the transfer to SPROC$^{\mathtt{dt}}$ in §4. Finally in §5 we translate signals into hyperstreams, and also the safety guarantee for hyperstreams (obtained by the SPROC$^{\mathtt{dt}}$ type system) to that for signals. §6 is devoted to a verification example. Our intention is to use the current framework for hybrid systems (as mentioned above)—definitions like Def. 5.1 are worked out so that it accommodates many common hybrid dynamics. Our leading example (Example 1.1, which is used in §6) is however a totally continuous one; this is due to the limited space. In §7 we conclude.

We defer most of the proofs to the extended version [31]. In this paper we sometimes refer to "Appendix"; it is found in [31].

Related work The current work shares with [4] the observation of the similarity between signal processing and stream processing. In [4] they extend Lustre by ODEs. They go on to a compilation framework that separates discrete and continuous parts of a program, passing the latter to an external solver to approximate continuous dynamics. For the correctness of the compilation they introduce NSA-based formal semantics [5], which like ours takes continuous dynamics as a succession of infinitesimal jumps. They also employ a type system for the separation of discrete and continuous parts of a program. Despite these similarities, the current work's objective is quite different from theirs—we aim to exploit NSA's logical infrastructure to transfer deductive (i.e. logical) verification from discrete stream processing to continuous-time signal processing. The extension of Lustre by ODEs in [5] is not designed towards this objective.

Formal verification of Simulink diagrams has been studied e.g. in [10, 28, 33]. In [28] Simulink diagrams are translated into hybrid automata, which are amenable to model checking. In [33] translation of a discrete fragment of Simulink into Lustre is presented. [10] combines symbolic analysis and numerical simulation, towards the goal of enhanced simulation coverage. All these papers agree on one point: Simulink lacks formal semantics. In [7, 10] Simulink semantics is defined "operationally" by formalizing the simulation algorithms used in the implementation of Simulink. We hope our hyperstream modeling will serve as a basis of denotational semantics of Simulink.

Turning to the purely discrete world, formal verification of stream processing systems is studied often in the abstract interpretation community [11]. Application of these results to our current deductive approach is an interesting direction of future work.

For hybrid systems in general, there have been extensive research efforts from the formal verification community. Unlike the current work where we turn flow into jump via dt, most of them feature acute distinction between flow- and jump-dynamics. These include: model-checking approaches based on hybrid automata [2]; deductive approaches, one of the most notable of which is a recent series of work by Platzer and his colleagues (including [21, 23]). Interestingly in [23] it is argued that being hybrid imposes no additional burden to deductive verification. This concurs with our NSA view that being discrete and being continuous/hybrid are "logically the same."

Some verification techniques from the static analysis community have been successfully used in hybrid applications (modeled with explicit differential equations) [17, 25–27]. The basic idea of the current work—also of our previous [13, 30]—is to transfer discrete verification techniques *as they are* to continuous/hybrid settings.

It is never our intention to champion the superiority of discrete techniques to continuous ones. The formal verification community has worked out a stock of discrete techniques; our case is that their application domain can be pushed further to continuous/hybrid. Indeed we see ODEs as an extremely efficient formalism for continuous dynamics. We plan to incorporate them into our NSA framework.

The use of NSA as a foundation of hybrid system modeling is not proposed for the first time; see e.g. [3, 5, 6]. Compared to this existing body of work, we claim our novelty is the use of NSA's logical infrastructure (especially the transfer principle) for deductive verification, based on a concrete modeling language.

In particular, the basic idea of the current paper (namely, stream processing + NSA = signal processing) as well as two important technical ideas (namely: infinitesimal sampling intervals and domain theory in NSA) are already in [3]. Unlike the current paper where we introduce a concrete programming (or modeling) language SPROC$^{\mathsf{dt}}$, in [3] they work with an abstract (graphical) language of string diagrams for monoidal categories.

Notations and terminology The syntactic equality is denoted by $\equiv$.

An infinite stream $s = (a_0, a_1, \dots)$ over S is identified with a function $s : \mathbb{N} \to S$. We write $s(i)$ for its i-th element, i.e. $s(i) = a_i$.

For a nonnegative real $r \in \mathbb{R}_{\geq 0}$, $\lceil r \rceil \in \mathbb{N}$ denotes the least natural number that is not smaller than r, that is, $r \leq \lceil r \rceil < r + 1$.

In this paper we use some domain theory, for which our principal reference is [1]. We will be using ω-cpo's—calling them simply "cpo's"—while in [1] most results are formulated in terms of directed cpo's. The equivalence between these two "cpo's" is found in [1, Prop. 2.1.15]. We also assume the least element $\perp$ in cpo's.

2. A Nonstandard Analysis Primer

Here we list the minimal set of necessary definitions and results in nonstandard analysis (NSA). More details are found e.g. in [12, 14].

2.1 Infinitesimals in NSA

First we present an elementary part of NSA. We fix an *index set* $I = \mathbb{N}$, and an *ultrafilter* $\mathcal{F} \subseteq \mathcal{P}(I)$ that extends the cofinite filter $\mathcal{F}_c := \{S \subseteq I \mid I \setminus S \text{ is finite}\}$. Its properties to be noted: 1) for any $S \subseteq I$, exactly one of S and $I \setminus S$ belongs to $\mathcal{F}$; 2) if S is *cofinite* (i.e. $I \setminus S$ is finite), then S belongs to $\mathcal{F}$.

Definition 2.1 (Hypernumber $d \in {}^*X$)**.** For a *base set* X (typically it is $\mathbb{N}$, $\mathbb{R}$ or $\mathbb{C}$), we define the set ${}^*X$ of *hypernumbers* by ${}^*X := X^I/{\sim_{\mathcal{F}}}$. It is the set of infinite sequences on X modulo the following equivalence $\sim_{\mathcal{F}}$: we define $(a_0, a_1, \dots) \sim_{\mathcal{F}} (a'_0, a'_1, \dots)$ by

$$\{i \in I \mid a_i = a'_i\} \in \mathcal{F} \quad , \tag{8}$$

for which we say "$a_i = a'_i$ for almost every i."

Therefore, given that two sequences $(a_i)_i$ and $(a'_i)_i$ coincide except for finitely many indices i, they represent the same hypernumber. The predicates other than $=$ (such as $<$) are defined in the same way. A notable consequence is the existence of an infinitesimal number: a hyperreal $\omega^{-1} := [\,(1, \frac{1}{2}, \frac{1}{3}, \dots)\,]$ is positive ($0 < \omega^{-1}$) but is smaller than any (standard) positive real $r = [(r, r, \dots)]$.

Definition 2.2 (Shadow)**.** A hyperreal r is *limited* if it is not infinite, i.e. if there is a standard positive real $K \in \mathbb{R}$ such that $-K < r < K$. It is well-known (see [12, 14]) that a limited hyperreal r has a unique standard real that is infinitely close to r. This standard real is called the *shadow* of r and denoted by $\mathrm{sh}(r)$.

The notion of shadow is a generalization of that of limit: if $(a_i)_i$ converges then $\mathrm{sh}\big([(a_0, a_1, \dots)]\big) = \lim_{i \to \infty} a_i$. See e.g. [12, 14].

Remark 2.3. It is common in NSA to take an index set I that is bigger than $\mathbb{N}$, and an ultrafilter $\mathcal{F} \subseteq \mathcal{P}(I)$ over I. The merit of doing so is that the resulting monomorphism ${}^*(\_)$ (§2.2) can be chosen to be an *enlargement*; see [14, Chap. II]. In this paper, however, we favor concreteness and choose $I = \mathbb{N}$ as the index set.

2.2 NSA in Superstructure

What we need from the logical machinery of NSA goes beyond the elementary fragment presented above. It employs a set theory-like formal language $\mathscr{L}_X$ and a so-called *superstructure* as a model. The definitions and results listed below are all well-established and commonly used in NSA. We follow [14, Chap. II], in which more details can be found.

Superstructure A superstructure is a "universe," constructed step by step from a certain base set X. We assume $\mathbb{N} \subseteq X$.

Definition 2.4 (Superstructure)**.** A *superstructure* $V(X)$ over X is:

$$\begin{array}{l} V(X) := \bigcup_{n \in \mathbb{N}} V_n(X) \quad , \quad \text{where} \\ V_0(X) := X \quad \text{and } V_{n+1}(X) := V_n(X) \cup \mathcal{P}(V_n(X)) \quad . \end{array}$$

(Ordered) pairs (a, b) and tuples $(a_1, \dots, a_m)$ are defined in $V(X)$ as is usually done in set theory, e.g. $(a, b) := \{\{a\}, \{a, b\}\}$. The set $V(X)$ is closed under many set formation operations. For example the function space $a \to b$ is thought of as a collection of special binary relations ($a \to b \subseteq \mathcal{P}(a \times b)$), hence is in $V(X)$.

****-Transform*** We use the following predicate logic $\mathscr{L}_X$.

Definition 2.5 (The language $\mathscr{L}_X$)**.** *Terms* in $\mathscr{L}_X$ consist of: variables $x, y, x_1, x_2, \dots$; and a constant a for each entity $a \in V(X)$.
Formulas in $\mathscr{L}_X$ are constructed as follows.

- The predicate symbols are $=$ and $\in$; both are binary. The *atomic formulas* are of the form $s = t$ or $s \in t$ (where s and t are terms).
- Any Boolean combination of formulas is a formula. We use the symbols $\wedge, \vee, \neg$ and $\Rightarrow$.
- Given a formula A, a variable x and a term s, the expressions $\forall x \in s.\, A$ and $\exists x \in s.\, A$ are formulas.

Note that quantifiers always come with a bound s. The language $\mathscr{L}_X$ depends on the choice of X (it determines the set of constants). We shall also use the following syntax sugars in $\mathscr{L}_X$, as is common in NSA. Their translation into proper $\mathscr{L}_X$ formulas is straightforward.

(s, t)	pair	$(s_1, \dots, s_m)$	tuple
$s \times t$	direct product		
$s \subseteq t$	inclusion, short for $\forall x \in s.\, x \in t$		
$s(t)$	function application; short for x s.t. $(t, x) \in s$		
$s \circ t$	function composition, $(s \circ t)(x) = s(t(x))$		
$s \leq t$	inequality in $\mathbb{N}$; short for $(s, t) \in {\leq}$ where ${\leq} \subseteq \mathbb{N}^2$		

Remark 2.6. We note that $\mathscr{L}_X$ resides on a different level from the languages that we introduce later, such as SPROC, SPROC$^{\mathsf{dt}}$ and their assertion languages. $\mathscr{L}_X$ is used to define the semantics of those object-level languages, and is a *meta language* in this sense.

Definition 2.7 (Semantics of $\mathscr{L}_X$)**.** We interpret $\mathscr{L}_X$ in the superstructure $V(X)$ in the obvious way. Let A be a closed formula; we say A is *valid* if A is true in $V(X)$.

Validity is defined only for closed formulas.

The so-called *ultrapower construction* yields a canonical map

$$^*(\_) \;:\; V(X) \longrightarrow V(^*X) \;, \qquad a \longmapsto {}^*a \tag{9}$$

that is called the **-transform*. It is a map from the universe $V(X)$ of standard entities to $V(^*X)$ of nonstandard entities. We skip the details of its construction; later in this section we take a closer look.

The map $^*(\_)$ becomes a *monomorphism*, a notion in NSA. Most notably it satisfies the *transfer principle* (Lem. 2.9).

Definition 2.8 (*-transform of formulas)**.** Let A be a formula in $\mathscr{L}_X$. The **-transform* of A, denoted by $^*A$, is a formula in $\mathscr{L}_{^*X}$ obtained by replacing each constant a occurring in A with the constant $^*a$ that designates the element $^*a \in V(^*X)$.

Lemma 2.9 (The transfer principle)**.** *For any closed formula A in $\mathscr{L}_X$, A is valid (in $V(X)$) if and only if $^*A$ is valid (in $V(^*X)$).* □

The transfer principle is a powerful result and we will totally rely on it in the semantics of SPROC$^{\mathrm{dt}}$. Here are the first examples of its use.

Lemma 2.10.
1. *For $a \in V(X) \setminus X$ we obtain an injective map*
$$^*(\_) \;:\; a \longrightarrow {}^*a \;, \quad (b \in a) \longmapsto (^*b \in {}^*a) \tag{10}$$
as a restriction of $^*(\_)$ in (9).
2. *If a is a finite set, the map (10) is an isomorphism $a \xrightarrow{\cong} {}^*a$.*
3. *Let $a \to b$ be the set of functions from a to b. We have $^*(a \to b) \subseteq {}^*a \to {}^*b$.*
4. *$^*(a_1 \times \cdots \times a_m) = {}^*a_1 \times \cdots \times {}^*a_m$; and $^*(a_1 \cup \cdots \cup a_m) = {}^*a_1 \cup \cdots \cup {}^*a_m$.*
5. *For a binary relation $r \subseteq a \times a$, we have $^*r \subseteq {}^*a \times {}^*a$. Moreover, r is an order if and only if $^*r$ is an order.* □

Internal Sets The distinction between *internal* and *external* entities is central in NSA. In this paper however it is much of formality, since all the entities we use are internal. Here we present only the relevant definitions, leaving their intuitions to [14, §II.6]. In Appendix B, especially Rem. B.8, we will see that being internal is crucial for transfer.

Definition 2.11 (Internal entity)**.** An element $b \in V(^*X)$ is *internal* with respect to $^*(\_) : V(X) \to V(^*X)$ if there is $a \in V(X)$ such that $b \in {}^*a$. It is *external* if it is not internal.

Lemma 2.12. *$f : {}^*a \to {}^*b$ is internal if and only if $f \in {}^*(a \to b)$.* □

The ultrapower construction We collect some necessary facts about the ultrapower construction of the monomorphism $^*(\_)$ in (9). Its details are beyond our scope; they are found in [14, §II.4].

The map $^*(\_)$ in fact factorizes into the following three steps.

$$\begin{array}{ccc} V(X) & \xrightarrow{\quad ^*(\_) \quad} & V(^*X) \\ {\scriptstyle \overline{(\_)}}\downarrow & & \uparrow{\scriptstyle M} \\ \bigcup_{n\in\mathbb{N}} \big(V_n(X) \setminus V_{n-1}(X)\big)^I & \xrightarrow[{[\_]}]{} & \prod^0_{\mathcal{F}} V(X) \end{array} \tag{11}$$

The first factor $\overline{(\_)}$ maps $a \in V(X)$ to the constant function $\overline{a}$ such that $\overline{a}(i) = a$ for each $i \in I$; recall that we have chosen $I = \mathbb{N}$ (Rem. 2.3). The second $[\_]$ takes a quotient modulo the ultrafilter $\mathcal{F}$; finally the third factor M is the so-called *Mostowski collapse*.

For an intuition let us exhibit these maps in the simple setting of §2.1. The first factor $\overline{(\_)}$ corresponds to forming constant streams: $a \mapsto \overline{a} = (a, a, \dots)$. The second $[\_]$ is quotienting modulo $\sim_{\mathcal{F}}$ of (8). The third map M does nothing—it is a book-keeping function that is only needed in the extended setting of superstructures.

The next result [14, Thm. 4.5] is about "starting from the lower-left corner" in (11). It follows from the definition of M and is a crucial step in the proof of the transfer principle (Lem. 2.9). It serves as an important lemma, too, later for the semantics of SPROC$^{\mathrm{dt}}$.

Lemma 2.13 (Łoś' theorem)**.** *Let A be a formula in $\mathscr{L}_X$ with its free variables contained in $\{x_1, \dots, x_m\}$; and $a_1, \dots, a_m \in \bigcup_{n\in\mathbb{N}} \big(V_n(X) \setminus V_{n-1}(X)\big)^I$. Then*

$$\begin{array}{l} ^*A\big[\, M[a_1]/x_1, \dots, M[a_m]/x_m \,\big] \text{ is valid} \\ \iff \quad \big\{i \in I \mid A[a_1(i)/x_1, \dots, a_m(i)/x_m] \text{ is valid}\big\} \in \mathcal{F} \;. \end{array}$$

As a special case, let $S \in V(X)$, then

$$M[a] \in {}^*S \quad\iff\quad a(i) \in S \ \text{ for almost every } i. \qquad \square$$

Corollary 2.14. *Let $a, b \in V(X)$; and for each $i \in I$, $f_i \in (a \to b)$ and $x_i \in a$. Then $M[(f_i)_{i\in I}]$ is an internal function $^*a \to {}^*b$; and $M[(x_i)_{i\in I}] \in {}^*a$. Moreover,*

$$M[(f_i(x_i))_{i\in I}] = \Big(M[(f_i)_{i\in I}]\Big)\Big(M[(x_i)_{i\in I}]\Big) \;. \qquad \square$$

3. The Stream Processing Language SPROC

In this section we introduce the language SPROC for stream processing, together with its denotational semantics and a type system. The last is much like a Hoare-style program logic for partial correctness. The whole framework is nothing surprising: SPROC is modeled after Lustre [9]; its semantics is defined as usual, following Kahn [15]; and the type system is rudimentary with a limited expressive power. The point is their clean logical foundations, which allow us to transfer the whole framework—via NSA—to SPROC$^{\mathrm{dt}}$ (§4).

3.1 SPROC: Syntax

For an example of an SPROC program, see Example 1.1. It is an SPROC$^{\mathrm{dt}}$ program, but the two languages are very close.

Definition 3.1 (SPROC)**.** We fix a set $\mathbf{SVar}$ of *stream variables* and, for each $m, n \in \mathbb{N}$, a set $\mathbf{NdName}_{m,n}$ of *node names* of arity (m, n). These sets are assumed to be disjoint. The syntax of SPROC is defined in Table 1. Some of its details are in order.

The set $\mathbf{SExp}_{\mathbb{C}}$ consists of the $\mathbb{C}$*-stream expressions*. A constant $c \in \mathbb{C}$ stands for the $\mathbb{C}$-stream $(c, c, \dots)$. The operator ^ is for the power: $(a_0, a_1, \dots) \mathbin{\hat{}} (b_0, b_1, \dots) = (a_0^{b_0}, a_1^{b_1}, \dots)$. For each $j \in \mathbb{N}$ we have an operator $\mathtt{fby}^j$ ("followed by"). It means

$$(a_0, a_1, \dots)\, \mathtt{fby}^j\, (b_0, b_1, \dots) \;=\; (a_0, \dots, a_{j-1}, b_0, b_1, \dots) \;. \tag{12}$$

The expression $\mathtt{proj}_k\, f(e_1, \dots, e_m)$ invokes the node whose name is f (declared elsewhere in the same program), feeds it with the input $(e_1, \dots, e_m)$, and returns the k-th component of its output.

The set $\mathbf{SExp}_{\mathbb{B}}$ consists of the expressions for streams in the Boolean values $\mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$. The operators $=$, $\mathtt{isReal}$ and $<$ are the obvious extensions of $= : \mathbb{C}^2 \to \mathbb{B}$, $\mathtt{isReal} : \mathbb{C} \to \mathbb{B}$ and $< : \mathbb{C}^2 \to \mathbb{B}$. The last is defined by

$$c_1 < c_2 \;:=\; \begin{cases} \mathrm{tt} & \text{if } c_1, c_2 \in \mathbb{R} \text{ and } c_1 < c_2, \\ \mathrm{ff} & \text{otherwise.} \end{cases} \tag{13}$$

Each *node* $\mathtt{nd} \in \mathbf{Nodes}$ comes with a certain arity (m, n) and its name f is chosen from $\mathbf{NdName}_{m,n}$. It takes m-many $\mathbb{C}$-streams as input and returns n-many $\mathbb{C}$-streams as output. In the node $\mathtt{nd}$ in Table 1, the variable x_i is for an input stream; and the local variable y_i is used in the (mutually) recursive computation inside the node (specified by $y_1 = e'_1; \ldots ; y_l = e'_l$). These x_i's and y_i's together constitute the set of *bound variables* in nd. The restriction in Table 1 dictates that only these variables are allowed to occur in nd.

Finally, a *program* of SPROC is a finite sequence of nodes, with the last one designated as the *main* node. The restriction in Table 1 means that we can invoke a node only if it is declared in the program.

3.2 SPROC: Denotational Semantics

We define the semantics of SPROC in the denotational style, exploiting the cpo structure of streams. This approach to the denotational semantics of stream processing systems dates back to Kahn [15]. Specifically, our semantic domains are as follows.

Definition 3.2 ($\mathbb{C}^\infty, \mathbb{B}^\infty$)**.** By $\mathbb{C}^\infty$ we denote the set of finite and infinite streams over $\mathbb{C}$. That is, $\mathbb{C}^\infty := \mathbb{C}^* \cup \mathbb{C}^{\mathbb{N}}$. We also define $\mathbb{B}^\infty$ for $\mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$ by $\mathbb{B}^\infty := \mathbb{B}^* \cup \mathbb{B}^{\mathbb{N}}$.

Notation 3.3. In what follows it is convenient to regard a finite stream as if it is an infinite stream. Using $\perp$ ("undefined"), we identify $(a_0, a_1, \ldots, a_m)$ with an infinite stream

$$(a_0, a_1, \ldots, a_m, \perp, \perp, \ldots) \ .$$

The intuition is: "production of the element a_{m+1} never terminated; thus the elements henceforth never got produced." Hence in $(a_n)_{n\in\mathbb{N}} \in \mathbb{C}^\infty$, if $a_m = \perp$ then $a_{m'} = \perp$ for any $m' \geq m$.

The following result underpins Kahn's approach [15].

Lemma 3.4. *The prefix order $\sqsubseteq$ on $\mathbb{C}^\infty$ makes it a cpo, that is, any ascending chain $s_0 \sqsubseteq s_1 \sqsubseteq \cdots$ has a supremum $\bigsqcup_i s_i$. Its least element is the empty stream ε. The same holds for $\mathbb{B}^\infty$ too.* □

Based on this observation, we introduce the semantics as follows.

Definition 3.5 (Variable/node environment)**.** A *(stream) variable environment* is a function $J : \mathbf{SVar} \to \mathbb{C}^\infty$ that assigns $J(x) \in \mathbb{C}^\infty$ to each stream variable x. A *node environment* K assigns, to each node name $f \in \mathbf{NdName}$, a continuous function

$$K(f) \ \in \ \big((\mathbb{C}^\infty)^{m_f} \to_{\mathrm{ct}} (\mathbb{C}^\infty)^{n_f} \big) \ . \tag{14}$$

Here (m_f, n_f) is the arity of f; and the set $(\mathbb{C}^\infty)^{m_f} \to_{\mathrm{ct}} (\mathbb{C}^\infty)^{n_f}$ is that of *continuous* (i.e. $\bigsqcup$-preserving, but not necessarily $\perp$-preserving) functions from $(\mathbb{C}^\infty)^{m_f}$ to $(\mathbb{C}^\infty)^{n_f}$, with respect to the order $\sqsubseteq$ in Lem. 3.4. Therefore K is an element of the following set.

$$K \ \in \ \textstyle\prod_{f\in\mathbf{NdName}} \big((\mathbb{C}^\infty)^{m_f} \to_{\mathrm{ct}} (\mathbb{C}^\infty)^{n_f} \big) \ . \tag{15}$$

We denote the sets of (stream) variable environments and node environments by $\mathbf{SVarEnv}$ and $\mathbf{NdEnv}$, respectively.

Lemma 3.6. *The sets $\mathbf{SVarEnv}$ and $\mathbf{NdEnv}$ are cpo's, with the pointwise extension of the order structure of $\mathbb{C}^\infty$. Specifically, between $K, K' \in \mathbf{NdEnv}$, we have $K \sqsubseteq K'$ if and only if*

$$\begin{aligned} &\forall m, n \in \mathbb{N}.\, \forall f \in \mathbf{NdName}_{m,n}.\, \forall \vec{s} \in (\mathbb{C}^\infty)^m. \\ &\qquad \forall k \in [1, n].\quad \pi_k\big(K(f)(\vec{s})\big) \ \sqsubseteq \ \pi_k\big(K'(f)(\vec{s})\big) \ . \end{aligned}$$

Here π_k is the k-th projection $\pi_k : (\mathbb{C}^\infty)^n \to \mathbb{C}^\infty$. The order on $\mathbf{SVarEnv}$ is similar. □

Using these environments we define the semantics of SPROC expressions as follows. We go step by step.

Definition 3.7 ($[\![e]\!]_{J,K}$ and $[\![b]\!]_{J,K}$)**.** In Table 2 we define the denotation $[\![e]\!]_{J,K} \in \mathbb{C}^\infty$ of a $\mathbb{C}$-stream expression $e \in \mathbf{SExp}_{\mathbb{C}}$, under variable and node environments J and K.

Here the definition of $[\![e_1 \ \mathtt{aop}\ e_2]\!]_{J,K}$ simply says that $\mathtt{aop}$ is applied elementwise. Recall that a finite stream $(a_0 \ldots a_m)$ is identified with $(a_0, \ldots, a_m, \perp, \perp, \ldots)$; the operator $\mathtt{aop}_\perp$ returns $\perp$ if any of its arguments is $\perp$. This is the same for $\mathtt{if}_\perp \ldots \mathtt{then} \ldots \mathtt{else} \ldots$, $\wedge_\perp$, $\neg_\perp$, etc. that appear later in Table 2. The definition of $[\![e_1 \ \mathtt{fby}^j\ e_2]\!]_{J,K}$ is the equation (12) put in formal terms. Recall that $\perp$ means "nontermination" (Notation 3.3). In the definition of $[\![\mathtt{proj}_k\, f(e_1, \ldots, e_m)]\!]_{J,K}$, recall that π_k is the k-th projection (see Lem. 3.6); also note the type of K (see (14)).

We simultaneously interpret $\mathbb{B}$-stream expressions, as in Table 2. Recall our definition of $<$ between complex numbers (see (13)).

The semantics of intra/inter-node recursion is by least fixed points.

Definition 3.8 ($[\![\mathsf{nd}]\!]_K$)**.** Let nd be a node

$$\mathsf{nd} :\equiv \left[\begin{array}{l} \mathtt{node}\ f(x_1, \ldots, x_m)\ \mathtt{returns}\ (e_1, \ldots, e_n) \\ \mathtt{where}\ y_1 = e'_1; \ldots ; y_l = e'_l \end{array} \right] \tag{16}$$

of arity (m, n). We define its denotation

$$[\![\mathsf{nd}]\!]_K \ : \ (\mathbb{C}^\infty)^m \longrightarrow (\mathbb{C}^\infty)^n \tag{17}$$

as follows. Given $\vec{s} = (s_1, \ldots, s_m) \in (\mathbb{C}^\infty)^m$ as input, first we solve the following recursive equation, and obtain a variable environment J_0 as its least solution.

$$J_0 \ = \ J_0 \left[\begin{array}{l} x_1 \mapsto s_1,\ \ldots,\ x_m \mapsto s_m, \\ y_1 \mapsto [\![e'_1]\!]_{J_0,K},\ \ldots,\ y_l \mapsto [\![e'_l]\!]_{J_0,K} \end{array} \right] \tag{18}$$

On the right-hand side, $[x_1 \mapsto s_1, \ldots]$ means a function update. The variable environment J_0 thus obtained is used in:

$$[\![\mathsf{nd}]\!]_K(s_1, \ldots, s_m) \ := \ (\, [\![e_1]\!]_{J_0,K},\ \ldots,\ [\![e_n]\!]_{J_0,K}\,) \in (\mathbb{C}^\infty)^n \ .$$

Definition 3.9 ($[\![\mathsf{pg}]\!]$)**.** Let pg be a program $[\mathsf{nd}_1, \ldots, \mathsf{nd}_N; \mathsf{nd}_{\mathsf{Main}}]$; $f_1, \ldots, f_N$ and f_{Main} be the names of $\mathsf{nd}_1, \ldots, \mathsf{nd}_N$ and $\mathsf{nd}_{\mathsf{Main}}$; and $(m_{\mathsf{Main}}, n_{\mathsf{Main}})$ be the arity of $\mathsf{nd}_{\mathsf{Main}}$. We define the denotation $[\![\mathsf{pg}]\!] : (\mathbb{C}^\infty)^{m_{\mathsf{Main}}} \to (\mathbb{C}^\infty)^{n_{\mathsf{Main}}}$ as follows. We define a node environment K_0 to be the least solution of the following recursive equation.

$$K_0 \ = \ K_0 \left[\begin{array}{l} f_1 \mapsto [\![\mathsf{nd}_1]\!]_{K_0},\ \ldots,\ f_N \mapsto [\![\mathsf{nd}_N]\!]_{K_0}, \\ f_{\mathsf{Main}} \mapsto [\![\mathsf{nd}_{\mathsf{Main}}]\!]_{K_0} \end{array} \right] \tag{19}$$

The node environment K_0 thus obtained is used in the following. (Note the type; see (17))

$$[\![\mathsf{pg}]\!] \ := \ [\![\mathsf{nd}_{\mathsf{Main}}]\!]_{K_0} \quad : (\mathbb{C}^\infty)^{m_{\mathsf{Main}}} \to (\mathbb{C}^\infty)^{n_{\mathsf{Main}}}$$

We need the following lemmas for Def. 3.8–3.9 to make sense. These follow from the fact that all the constructs in the denotational semantics are continuous, which is proved in Appendix A.

Lemma 3.10. *For any node nd and any $K \in \mathbf{NdEnv}$, the function $[\![\mathsf{nd}]\!]_K : (\mathbb{C}^\infty)^m \to (\mathbb{C}^\infty)^n$ is continuous. Therefore the function on the right-hand side of (19) is indeed a node environment.* □

Lemma 3.11. *The recursive equations (18–19) have least solutions.*

Proof. By the continuity of the relevant operations, including

$$\begin{aligned} \Phi :\ \ & \mathbf{SVarEnv} \longrightarrow \mathbf{SVarEnv}\ , \\ & J \longmapsto J \left[\begin{array}{l} x_1 \mapsto s_1,\ \ldots,\ x_m \mapsto s_m, \\ y_1 \mapsto [\![e'_1]\!]_{J,K},\ \ldots,\ y_l \mapsto [\![e'_l]\!]_{J,K} \end{array} \right] \ , \end{aligned} \tag{20}$$

and Lem. 3.6. □

3.3 SPROC: Type System for Safety Guarantee

We now present a "program logic" for SPROC. SPROC is a first-order functional language; therefore, as usual, our logic takes the form of a *type system*. There we identify types with predicates.

Our type system is rather restricted and is aimed (solely) at the *partial* guarantee of safety, that is, it gives no guarantee in the

$$
\begin{array}{rcl}
\mathbf{SExp}_{\mathbb{C}} \ni e & ::= & x \mid c \mid e_1 + e_2 \mid e_1 \times e_2 \mid e_1 \mathbin{\hat{}} e_2 \mid e_1 \,\texttt{fby}^j\, e_2 \mid \texttt{if}\ b\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{proj}_k\, f(e_1, \dots, e_m) \\
 & & \quad \text{where } x \in \mathbf{SVar};\ c \in \mathbb{C};\ j \in \mathbb{N};\ b \in \mathbf{SExp}_{\mathbb{B}};\ f \in \mathbf{NdName}_{m,n};\text{ and } k \in [1,n] \\
\mathbf{SExp}_{\mathbb{B}} \ni b & ::= & \texttt{true} \mid \texttt{false} \mid b_1 \wedge b_2 \mid \neg b \mid e_1 = e_2 \mid \texttt{isReal}(e) \mid e_1 < e_2 \qquad \text{where } e, e_i \in \mathbf{SExp}_{\mathbb{C}} \\
\mathbf{Nodes} \ni \mathsf{nd} & ::= & \left[\begin{array}{l} \texttt{node}\ f(x_1, \dots, x_m)\ \texttt{returns}\ (e_1, \dots, e_n) \\ \texttt{where}\ y_1 = e'_1; y_2 = e'_2; \dots; y_l = e'_l \end{array}\right] \quad \begin{array}{l} \text{where } f \in \mathbf{NdName}_{m,n};\ x_i, y_i \in \mathbf{SVar};\ e_i, e'_i \in \mathbf{SExp}_{\mathbb{C}}; \\ x_1, \dots, x_m, y_1, \dots, y_n \text{ are all distinct; and the variables occurring} \\ \text{in } e_i, e'_i \text{ are restricted to } x_i \text{ and } y_i \end{array} \\
\mathbf{Programs} \ni \mathsf{pg} & ::= & [\mathsf{nd}_1, \mathsf{nd}_2, \dots, \mathsf{nd}_m; \mathsf{nd}_{\mathsf{Main}}] \quad \begin{array}{l} \text{where } \mathsf{nd}_i, \mathsf{nd}_{\mathsf{Main}} \in \mathbf{Nodes}; \text{ and the node names occurring in } \mathsf{nd}_i \text{ or } \mathsf{nd}_{\mathsf{Main}} \\ \text{are restricted to } f_1, \dots, f_m \text{ and } f_{\mathsf{Main}}, \text{ the (distinct) names of } \mathsf{nd}_1, \dots, \mathsf{nd}_m \text{ and } \mathsf{nd}_{\mathsf{Main}} \end{array}
\end{array}
$$

Table 1. Syntax of SPROC

$$
\begin{array}{l}
[\![x]\!]_{J,K} := J(x) \qquad [\![c]\!]_{J,K} := (c, c, \dots) \qquad [\![e_1 \,\texttt{aop}\, e_2]\!]_{J,K} := \big([\![e_1]\!]_{J,K}(n) \,\texttt{aop}_{\perp}\, [\![e_2]\!]_{J,K}(n) \big)_{n \in \mathbb{N}} \quad \text{where } \texttt{aop} \in \{+, \times, \hat{}\,\} \\
[\![e_1 \,\texttt{fby}^j\, e_2]\!]_{J,K} := \begin{cases} \big([\![e_1]\!]_{J,K}(0), [\![e_1]\!]_{J,K}(1), \dots, [\![e_1]\!]_{J,K}(j-1), [\![e_2]\!]_{J,K}(0), [\![e_2]\!]_{J,K}(1), \dots \big) & \text{if the length of } [\![e_1]\!]_{J,K} \text{ is at least } j \\ \big([\![e_1]\!]_{J,K}(0), [\![e_1]\!]_{J,K}(1), \dots, [\![e_1]\!]_{J,K}(k-1), \perp, \perp \dots \big) & \text{if the length of } [\![e_1]\!]_{J,K} \text{ is } k \text{ and } k < j \end{cases} \\
[\![\texttt{if}\ b\ \texttt{then}\ e_1\ \texttt{else}\ e_2]\!]_{J,K} := \big(\texttt{if}_{\perp}\ [\![b]\!]_{J,K}(n)\ \texttt{then}\ [\![e_1]\!]_{J,K}(n)\ \texttt{else}\ [\![e_2]\!]_{J,K}(n) \big)_{n \in \mathbb{N}} \\
[\![\texttt{proj}_k\, f(e_1, \dots, e_m)]\!]_{J,K} := \pi_k \Big(K(f)\big([\![e_1]\!]_{J,K}, \dots, [\![e_m]\!]_{J,K} \big) \Big) \\
[\![\texttt{true}]\!]_{J,K} := (\mathtt{tt}, \mathtt{tt}, \dots) \quad [\![\texttt{false}]\!]_{J,K} := (\mathtt{ff}, \mathtt{ff}, \dots) \quad [\![b_1 \wedge b_2]\!]_{J,K} := \big([\![b_1]\!]_{J,K}(n) \wedge_{\perp} [\![b_2]\!]_{J,K}(n) \big)_{n \in \mathbb{N}} \quad [\![\neg b]\!]_{J,K} := \big(\neg_{\perp} [\![b]\!]_{J,K}(n) \big)_{n \in \mathbb{N}} \\
[\![e_1 = e_2]\!]_{J,K} := \big([\![e_1]\!]_{J,K}(n) =_{\perp} [\![e_2]\!]_{J,K}(n) \big)_{n \in \mathbb{N}} \qquad [\![\texttt{isReal}(e)]\!]_{J,K} := \Big(\texttt{isReal}_{\perp}\big([\![e]\!]_{J,K}(n) \big) \Big)_{n \in \mathbb{N}} \\
[\![e_1 < e_2]\!]_{J,K}(n) := \begin{cases} [\![e_1]\!]_{J,K}(n) <_{\perp} [\![e_2]\!]_{J,K}(n) & \text{if } n = 0 \text{ or } [\![e_1 < e_2]\!]_{J,K}(n-1) \neq \perp \\ \perp & \text{if } [\![e_1 < e_2]\!]_{J,K}(n-1) = \perp \end{cases}
\end{array}
$$

Table 2. Denotation $[\![e]\!]_{J,K}$, $[\![b]\!]_{J,K}$

case of nontermination. Our focus on partial safety is influenced by [18]; and is much like common Hoare-style program logics. We leave as future work verification of liveness properties; the latter necessarily involves the analysis of termination (i.e. *productivity* in stream processing).

Remark 3.12. For functional stream processing languages, Nakano's type system [20] with the $\bullet$ modality is well-known. Its concern is for the productivity (i.e. totality, termination) of stream computation; this is orthogonal to ours (partial safety). In [16] semantics of stream processing subject to Nakano's types is proposed using ultrametric spaces—as an alternative to the Kahn-style cpo semantics which we have used—with its merit being that one can form a semantic domain consisting solely of *total* streams (i.e. $\mathbb{C}^{\mathbb{N}}$ instead of $\mathbb{C}^{\infty}$). Application of these results to SPROC, and further to SPROC$^{\mathrm{dt}}$, is left as future work.

3.3.1 SPROC: Type Syntax

Our type syntax is borrowed from that of *dependent type systems*. The latter are known for their expressiveness and have been used for verification of higher-order programs (e.g. in [32]). Our type system, as a feasibility study of the methodology, is much more restricted. See Example 3.14.

$$
\begin{array}{rcl}
\mathbf{AExp} \ni a & ::= & v \mid c \mid a_1 + a_2 \mid a_1 \times a_2 \mid a_1 \mathbin{\hat{}} a_2 \mid \lceil a_1 \rceil \\
 & & \quad \text{where } v \in \mathbf{Var} \text{ and } c \in \mathbb{C} \\
\mathbf{Fml} \ni P & ::= & \texttt{true} \mid \texttt{false} \mid P_1 \wedge P_2 \mid P_1 \vee P_2 \mid \neg P \mid \\
 & & a_1 = a_2 \mid \texttt{isReal}(a) \mid a_1 < a_2 \mid a_1 \leq a_2 \mid \\
 & & \forall v \in \mathbb{N}.\, P \mid \forall v \in \mathbb{C}.\, P \\
 & & \quad \text{where } v \in \mathbf{Var} \text{ and } a, a_i \in \mathbf{SExp}_{\mathbb{C}} \\
\mathbf{SType}_{\mathbb{C}} \ni \tau & ::= & \prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P\} \quad \text{where } u, v \in \mathbf{Var}, \\
 & & P \in \mathbf{Fml} \text{ and } \mathrm{FV}(P) \subseteq \{u, v\} \\
\mathbf{SType}_{\mathbb{B}} \ni \beta & ::= & \prod_{v \in \mathbb{N}} P \quad \text{where } v \in \mathbf{Var}, \\
 & & P \in \mathbf{Fml} \text{ and } \mathrm{FV}(P) \subseteq \{v\} \\
\mathbf{NdType}_{m,n} \ni \nu & ::= & (\tau_1, \dots, \tau_m) \to (\tau'_1, \dots, \tau'_n) \\
 & & \quad \text{where } \tau_i, \tau'_i \in \mathbf{SType}_{\mathbb{C}}
\end{array}
$$

Table 3. Type Syntax for SPROC

Definition 3.13 (Types for SPROC)**.** The syntax of our type system for SPROC is shown in Table 3.

The set $\mathbf{AExp}$ is that of *arithmetic expressions*, each of which denotes a number in $\mathbb{C}$. We assume a countable set $\mathbf{Var}$ of variables; note that this is different from the set $\mathbf{SVar}$ of *stream* variables. The *rounding up* operation $\lceil \_ \rceil$ (see §1) is included for a later use. The set $\mathbf{Fml}$ is that of *assertion formulas*; it follows the usual syntax of first-order predicate logic.

A *type* $\tau \in \mathbf{SType}_{\mathbb{C}}$ *for* $\mathbb{C}$*-streams* is an expression $\prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P\}$. It consists of variables u, v, a formula P, and the delimiter $\prod_{\_ \in \mathbb{N}} \{\_ \in \mathbb{C} \mid \_\}$. Its informal meaning is

$$
\{u \in \mathbb{C} \mid P[0/v]\} \times \{u \in \mathbb{C} \mid P[1/v]\} \times \{u \in \mathbb{C} \mid P[2/v]\} \times \cdots \ ;
$$

that is, the set of streams s such that its n-th element $u = s(n)$ and $v = n$ satisfy P, for each $n \in \mathbb{N}$. A *type* $\beta \in \mathbf{SType}_{\mathbb{B}}$ *for* $\mathbb{B}$*-streams* is similar: $t \models \prod_{v \in \mathbb{N}} P$ if $t(n)$ is equivalent to P, with $v = n$, for each $n \in \mathbb{N}$.

A *node type* $\nu \equiv (\tau_1, \dots, \tau_m) \to (\tau'_1, \dots, \tau'_n) \in \mathbf{NdType}_{m,n}$ represents the set of nodes of arity (m, n) that, when fed with streams satisfying $\tau_1, \dots, \tau_m$, output streams satisfying $\tau'_1, \dots, \tau'_n$.

In the expression $\prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P\} \in \mathbf{SType}_{\mathbb{C}}$, the variables u and v are *bound*. We identify types modulo renaming of these bound variables. The same is true of v in $\prod_{v \in \mathbb{N}} P \in \mathbf{SType}_{\mathbb{C}}$.

Example 3.14. The $\mathbb{C}$-stream type $\prod_{v \in \mathbb{N}} \{u \mid v \geq 3 \Rightarrow u \leq 1\}$ specifies that the elements $s(3), s(4), \dots$ of a stream s are real and ≤ 1. Our types can thus express rudimentary safety properties.

Regarding the limitation of the expressive power, it is straightforward to extend the type system with stream types of arity $k > 1$: $\prod_{v \in \mathbb{N}} \big\{ (u_1, \dots, u_k) \,\big|\, P \big\}$, where $\mathrm{FV}(P) \subseteq \{u_1, \dots, u_k, v\}$. This extension allows us to speak about correlations among distinct streams. So can we about correlations between input and output: we can prepare auxiliary output streams that copy input streams, and compare them with the output. Furthermore we can express temporal properties: to see if a stream s is increasing, we can check if the pair $(0 \,\texttt{fby}^1\, s, s)$ satisfies the binary type $\prod_{v \in \mathbb{N}} \{(u_1, u_2) \mid u_1 < u_2\}$. In this paper we restrict the presentation to unary stream types for the sake of simplicity.

This is not to say that the type system is amply expressive. For example, the type judgment $\Delta; x : \tau, y : \sigma \vdash \texttt{if}\ x = y\ \texttt{then}\ y\ \texttt{else}\ x : \tau$ (whose validity is not hard to see) cannot be derived by the typing rules. In its derivation one would need a $\mathbb{B}$-stream type $\prod_{v\in\mathbb{N}}\{x(v) = y(v)\}$, which is prohibited due to the free variables x, y in it.

Anyway, the syntactic restrictions in Table 3—compared to a fully-fledged dependent type system—simplify the type system drastically. For example, we do not need the well-formedness condition of type environments, which is usually needed in dependent type systems (see e.g. [32]). Relaxing these restrictions is future work.

3.3.2 SPROC: Type Semantics

Definition 3.15 (Valuation)**.** A *valuation* is either $\perp$ ("undefined") or a function $L : \mathbf{Var} \to \mathbb{C}$. The set of valuations is denoted by $\mathbf{Val}$, that is, $\mathbf{Val} = (\mathbf{Var} \to \mathbb{C}) \cup \{\perp\}$.

The function update $L[u_1 \mapsto c_1, \dots, u_m \mapsto c_m]$, with $L \in \mathbf{Val}$, $u_i \in \mathbf{Var}$ and $c_i \in \mathbb{C} \cup \{\perp\}$, is defined by:

$$L[\vec{u} \mapsto \vec{c}] \;:=\; \begin{cases} \perp & \text{if any of } c_i \text{ is } \perp \\ \text{(the usual function update)} & \text{otherwise.} \end{cases} \tag{21}$$

Therefore: if the length of $s \in \mathbb{C}^\infty$ is not more than n, valuation $L[u \mapsto s(n)]$ is defined to be $\perp$.

Definition 3.16 (Semantics of $\mathbf{AExp}$, $\mathbf{Fml}$)**.** The *denotation* $[\![a]\!]_L \in \mathbb{C} \cup \{\perp\}$, of an arithmetic expression $a \in \mathbf{AExp}$ under a valuation $L \in \mathbf{Val}$, is defined in the usual manner. We define $\models$ between valuations and formulas in the usual manner, too. For example,

$$\begin{aligned} L \models \mathtt{isReal}(a) &\overset{\text{def.}}{\iff} L = \perp \text{ or } [\![a]\!]_L \in \mathbb{R} \;, \\ L \models \forall v \in \mathbb{N}.P &\overset{\text{def.}}{\iff} L = \perp \text{ or } L[v \mapsto n] \models P \text{ for any } n \in \mathbb{N} \;, \end{aligned}$$

and so on. In particular, the valuation $\perp \in \mathbf{Val}$ satisfies any formula.

A formula P is *valid* ($\models P$) if $L \models P$ for any $L \in \mathbf{Val}$.

Definition 3.17 (Semantics of types)**.** Between a $\mathbb{C}$-stream $s \in \mathbb{C}^\infty$ and a $\mathbb{C}$-stream type $\tau \in \mathbf{SType}_{\mathbb{C}}$, $s \models \tau$ is defined by

$$\begin{aligned} &s \models \textstyle\prod_{v\in\mathbb{N}}\{u \in \mathbb{C} \mid P\} \overset{\text{def.}}{\iff} \\ &\quad L[v \mapsto n, u \mapsto s(n)] \models P \text{ for each } n \in \mathbb{N} \text{ and } L \in \mathbf{Val}. \end{aligned}$$

Note that the valuation L in the definition is vacuous, because of the restriction that $\mathrm{FV}(P) \subseteq \{u, v\}$. Note also that when $s(n) = \perp$, then $L[v \mapsto n, u \mapsto s(n)]$ is $\perp$ (Def. 3.15), which satisfies P. This reflects our focus on *partial correctness*: when the computation does not terminate—i.e. when the length of $s \in \mathbb{C}^\infty$ is l, and its $(l + 1)$-th element never gets produced—it does not matter what the type τ specifies about the $(l + 1)$-th and later elements of s.

Similarly, between a $\mathbb{B}$-stream $t \in \mathbb{B}^\infty$ and a $\mathbb{B}$-stream type $\beta \in \mathbf{SType}_{\mathbb{B}}$, the *satisfaction relation* $t \models \beta$ is defined as follows.

$$t \models \textstyle\prod_{v\in\mathbb{N}} P \overset{\text{def.}}{\iff} \begin{array}{l} \text{for each } n \in \mathbb{N},\, t(n) = \perp \text{ or} \\ \big(\, t(n) = \mathtt{tt} \;\Leftrightarrow\; L[v \mapsto n] \models P \text{ for each } L \in \mathbf{Val} \big) \end{array}$$

That is, "$t(n)$ if and only if $P[n/v]$."

Finally, between a continuous function $g : (\mathbb{C}^\infty)^m \to_{\mathrm{ct}} (\mathbb{C}^\infty)^n$ and a node type $\nu \in \mathbf{NdType}_{m,n}$, the *satisfaction relation* $\models$ is:

$$\begin{aligned} &g \models (\tau_1, \dots, \tau_m) \to (\tau'_1, \dots, \tau'_n) \overset{\text{def.}}{\iff} \\ &\forall s_1, \dots, s_m, s'_1, \dots, s'_n \in \mathbb{C}^\infty. \\ &\left[\begin{array}{l} s_1 \models \tau_1 \;\wedge \cdots \wedge\; s_m \models \tau_m \;\wedge\; (s'_1, \dots, s'_n) = g(s_1, \dots, s_m) \\ \quad\Longrightarrow\quad s'_1 \models \tau'_1 \;\wedge \cdots \wedge\; s'_n \models \tau'_n \end{array} \right] \end{aligned}$$

3.3.3 SPROC: Type Derivation

In type judgments we have two kinds of environments.

Definition 3.18 (Type environment)**.** A *stream type environment* $\Gamma = \{x_1 : \tau_1, \dots, x_m : \tau_m\}$ is a finite set of pairs of a stream variable $x_i \in \mathbf{SVar}$ and a $\mathbb{C}$-stream type $\tau_i \in \mathbf{SType}_{\mathbb{C}}$. We require $x_1, \dots, x_m$ to be distinct.

Similarly, a *node type environment* $\Delta = \{f_1 : \nu_1, \dots, f_m : \nu_m\}$ is a finite set, where $f_i \in \mathbf{NdName}$ is a node name and $\nu \in \mathbf{NdType}$ is a node type with the same arity.

We denote the sets of stream and node type environments by $\mathbf{STEnv}$ and $\mathbf{NdTEnv}$, respectively.

Notation 3.19. We sometimes write $\Gamma(x)$. In this case it is assumed that $x : \tau$ is in Γ with some τ; and $\Gamma(x)$ denotes this (unique) τ.

Definition 3.20 (Type judgment)**.** In our type system for SPROC we have four classes of type judgments. Here $\vdash$ is a (mere) delimiter.

- $\Delta; \Gamma \vdash e : \tau$, meaning: the $\mathbb{C}$-stream expression e is of the type τ if the variables denote the streams conforming to Γ and the node names denote the nodes conforming to Δ.
- $\Delta; \Gamma \vdash b : \tau_b$, meaning the same, between $\mathbb{B}$-stream expressions and $\mathbb{B}$-stream types.
- $\Delta \vdash \mathsf{nd} : \nu$, meaning: the node nd is of the node type ν if the node names denote the nodes that conform to Δ.
- $\vdash \mathsf{pg} : \nu$, meaning: pg's main node $\mathsf{nd}_{\mathsf{Main}}$ is of the node type ν.

Definition 3.21 (Type derivation)**.** The typing rules for SPROC are as shown in Table 4. We write $\Vdash \mathcal{J}$ if the type judgment $\mathcal{J}$ is derivable.

3.3.4 SPROC: Type Soundness

Definition 3.22. Between a stream variable environment $J \in \mathbf{SVarEnv} = (\mathbf{SVar} \to \mathbb{C}^\infty)$ and a stream type environment $\Gamma = \{x_1 : \tau_1, \dots, x_m : \tau_m\} \in \mathbf{STEnv}$, we define $J \models \Gamma$ by

$$J \models \Gamma \quad\overset{\text{def.}}{\iff}\quad J(x_i) \models \tau_i \text{ for each } i \in [1, m].$$

Here the latter $\models$ is as defined in Def. 3.17.

Similarly, between a node environment $K \in \mathbf{NdEnv}$ and a node type environment $\Delta = \{f_1 : \nu_1, \dots, f_m : \nu_m\} \in \mathbf{NdTEnv}$, we define $K \models \Delta$ if and only if $K(f_i) \models \nu_i$ for each $i \in [1, m]$.

Lemma 3.23. *The $\mathbb{C}$-stream $\perp \in \mathbb{C}^\infty$ (i.e. the empty stream) satisfies any type τ, that is, $\perp \models \tau$. The same for $\perp \in \mathbb{B}^\infty$. Similarly we have $\perp \models \Gamma$ for $\perp \in \mathbf{SVarEnv}$; and $\perp \models \Delta$ for $\perp \in \mathbf{NdEnv}$.* ☐

Definition 3.24 (Validity of type judgments)**.** We say a type judgment $\Delta; \Gamma \vdash e : \tau$ is *valid*, and write $\models \;\Delta; \Gamma \vdash e : \tau$, if for any $J \in \mathbf{SVarEnv}$ and $K \in \mathbf{NdEnv}$, $J \models \Gamma$ and $K \models \Delta$ imply $[\![e]\!]_{J,K} \models \tau$. The *validity* of the other three classes of type judgments is defined in the same manner.

Theorem 3.25 (Type soundness)**.** *A derivable type judgment is valid, that is, $\Vdash \mathcal{J}$ implies $\models \mathcal{J}$.*

Proof. The proof is mostly straightforward by induction. In Appendix D we show some exemplary cases. The cases (NODE) and (PROG) involve the principle of fixed point induction. ☐

4. The Hyperstream Processing Language SPROC$^{\mathrm{dt}}$

4.1 SPROC$^{\mathrm{dt}}$: Syntax

Definition 4.1 (SPROC$^{\mathrm{dt}}$)**.** The syntax of SPROC$^{\mathrm{dt}}$ is the same as that of SPROC (Table 1), except that we have two additional constructs—$\mathtt{dt}$ and $\mathtt{fby}^{\frac{r}{\mathtt{dt}}}$—in the set $\mathbf{SExp}_{\mathbb{C}}$.

$$\begin{aligned} \mathbf{SExp}_{\mathbb{C}} \ni e \;::=\;& x \mid c \mid e_1 + e_2 \mid \cdots \text{ (The same as in Table 1)} \\ & \mid \mathtt{dt} \mid e_1 \,\mathtt{fby}^{\frac{r}{\mathtt{dt}}}\, e_2 \qquad \text{where } r \in \mathbb{R}_{\geq 0} \end{aligned}$$

In SPROC$^{\mathrm{dt}}$ we call the elements of $\mathbf{SExp}_{\mathbb{C}}$ $\mathbb{C}$-*hyperstream expressions*. The same for $\mathbf{SExp}_{\mathbb{B}}$, too. Intuitively, the stream expression

$$\frac{}{\Delta;\Gamma\vdash x:\Gamma(x)}\ (\textsc{SVar}) \qquad \frac{}{\Delta;\Gamma\vdash c:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid u=c\}}\ (\textsc{Const})$$

$$\frac{\Delta;\Gamma\vdash e_i:\prod_{v\in\mathbb{N}}\{u_i\in\mathbb{C}\mid P_i\}\text{ for }i=1,2 \quad \models\forall v\in\mathbb{N}.\forall u_1,u_2,u\in\mathbb{C}.(P_1\wedge P_2\wedge u=(u_1\ \mathtt{aop}\ u_2)\ \Rightarrow P)}{\Delta;\Gamma\vdash e_1\ \mathtt{aop}\ e_2:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P\}}\ (\textsc{Aop})\quad (\mathtt{aop}\in\{+,\times,\hat{}\})$$

$$\frac{\Delta;\Gamma\vdash e_i:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P_i\}\text{ for }i=1,2 \quad \models\forall v\in\mathbb{N}.\forall u\in\mathbb{C}.\big((v<j\wedge P_1\ \Rightarrow P)\wedge(v\geq j\wedge P_2[v-j/v]\ \Rightarrow P)\big)}{\Delta;\Gamma\vdash e_1\ \mathtt{fby}^j\ e_2:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P\}}\ (\textsc{Fby}^j)$$

$$\frac{\Delta;\Gamma\vdash b:\prod_{v\in\mathbb{N}}P_b \quad \Delta;\Gamma\vdash e_i:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P_i\}\text{ for }i=1,2 \quad \models\forall v\in\mathbb{N}.\forall u\in\mathbb{C}.(P_b\wedge P_1\Rightarrow P)\wedge(P_b\wedge P_2\Rightarrow P)}{\Delta;\Gamma\vdash\mathtt{if}\ b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P\}}\ (\textsc{If})$$

$$\frac{\Delta;\Gamma\vdash e_i:\tau_i\text{ for }i\in[1,m] \quad \Delta(f)=(\tau_1,\dots,\tau_m)\to(\tau'_1,\dots,\tau'_n)}{\Delta;\Gamma\vdash\mathtt{proj}_k\ f(e_1,\dots,e_m):\tau'_k}\ (\textsc{NdCall})$$

$$\frac{\Delta;\Gamma\vdash e:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P'\} \quad \models\forall v\in\mathbb{N}.\forall u\in\mathbb{C}.\,P'\Rightarrow P}{\Delta;\Gamma\vdash e:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P\}}\ (\textsc{CStConseq})$$

$$\frac{\Delta;\Gamma\vdash b_i:\prod_{v\in\mathbb{N}}P_i\text{ for }i=1,2}{\Delta;\Gamma\vdash b_1\wedge b_2:\prod_{v\in\mathbb{N}}(P_1\wedge P_2)\text{ for }i=1,2}\ (\textsc{And}) \qquad (\textsc{True}),(\textsc{False}),(\textsc{Neg})\text{ are similar}$$

$$\frac{\Delta;\Gamma\vdash e_i:\prod_{v\in\mathbb{N}}\{u_i\in\mathbb{C}\mid P_i\}\text{ for }i=1,2 \quad \models\forall v\in\mathbb{N}.\forall u_1,u_2\in\mathbb{C}.\big(P_1\wedge P_2\Rightarrow(P\Leftrightarrow u_1=u_2)\big)}{\Delta;\Gamma\vdash e_1=e_2:\prod_{v\in\mathbb{N}}P}\ (\textsc{Equal})$$

$$\frac{\Delta;\Gamma\vdash e:\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P'\} \quad \models\forall v\in\mathbb{N}.\forall u\in\mathbb{C}.\big(P'\ \Rightarrow(P\Leftrightarrow\mathtt{isReal}(u))\big)}{\Delta;\Gamma\vdash\mathtt{isReal}(e):\prod_{v\in\mathbb{N}}P}\ (\textsc{IsReal})$$

$$\frac{\Delta;\Gamma\vdash e_i:\prod_{v\in\mathbb{N}}\{u_i\in\mathbb{C}\mid P_i\}\text{ for }i=1,2 \quad \models\forall v\in\mathbb{N}.\forall u_1,u_2\in\mathbb{C}.\big(P_1\wedge P_2\ \Rightarrow(P\Leftrightarrow(\mathtt{isReal}(u_1)\wedge\mathtt{isReal}(u_2)\wedge u_1<u_2))\big)}{\Delta;\Gamma\vdash e_1<e_2:\prod_{v\in\mathbb{N}}P}\ (\textsc{Less})$$

$$\frac{\Delta;\Gamma\vdash b:\prod_{v\in\mathbb{N}}P' \quad \models\forall v\in\mathbb{N}.\,P'\Leftrightarrow P}{\Delta;\Gamma\vdash b:\prod_{v\in\mathbb{N}}P}\ (\textsc{BStConseq})$$

$$\frac{\Gamma(x_i)\equiv\tau_i\text{ for }i\in[1,m] \quad \Delta;\Gamma\vdash e'_j:\Gamma(y_j)\text{ for }j\in[1,l] \quad \Delta;\Gamma\vdash e_k:\tau''_k\text{ for }k\in[1,n]}{\Delta\vdash\left[\begin{array}{l}\mathtt{node}\ f(x_1,\dots,x_m)\ \mathtt{returns}\ (e_1,\dots,e_n)\\ \mathtt{where}\ y_1=e'_1;\dots;y_l=e'_l\end{array}\right]:(\tau_1,\dots,\tau_m)\to(\tau''_1,\dots,\tau''_n)}\ (\textsc{Node})$$

$$\frac{\begin{array}{l}\Delta\vdash\mathsf{nd}:(\tau'_1,\dots,\tau'_m)\to(\sigma'_1,\dots,\sigma'_n)\\ \models\forall v\in\mathbb{N}.\,\forall u\in\mathbb{C}.\,P_i\Rightarrow P'_i\text{ for }i\in[1,m] \qquad \models\forall v\in\mathbb{N}.\,\forall u\in\mathbb{C}.\,Q'_j\Rightarrow Q_j\text{ for }j\in[1,n]\\ \big(\text{where }\tau_i\equiv\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P_i\},\ \tau'_i\equiv\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid P'_i\},\ \sigma_j\equiv\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid Q_j\},\ \sigma'_j\equiv\prod_{v\in\mathbb{N}}\{u\in\mathbb{C}\mid Q'_j\}\big)\end{array}}{\Delta\vdash\mathsf{nd}:(\tau_1,\dots,\tau_m)\to(\sigma_1,\dots,\sigma_n)}\ (\textsc{NdConseq})$$

$$\frac{\Delta\vdash\mathsf{nd}_i:\Delta(f_i)\text{ for }i\in[1,m] \quad \Delta\vdash\mathsf{nd}_{\mathsf{Main}}:\nu}{\vdash[\mathsf{nd}_1,\dots,\mathsf{nd}_m;\mathsf{nd}_{\mathsf{Main}}]:\nu}\ (\textsc{Prog}) \quad (f_i\text{ is the name of the nodes }\mathsf{nd}_i)$$

Table 4. Typing rules for SPROC

dt represents the constant stream $(\mathtt{dt},\mathtt{dt},\dots)$; each dt therein is thought of as a positive infinitesimal *sampling interval*. In addition to $\mathtt{fby}^j$ for the delay by j steps, we now have $\mathtt{fby}^{\frac{r}{\mathtt{dt}}}$ for delay by infinite steps. With dt being the sampling interval, $1/\mathtt{dt}$ is the *sampling frequency*. Therefore delay by $\frac{r}{\mathtt{dt}}$ steps means delay "by r seconds."

For the semantics of $\text{SPROC}^{\mathtt{dt}}$ we use the second key idea of *sectionwise execution* (see §1). In it an $\text{SPROC}^{\mathtt{dt}}$ program is first split up into its *sections*; each section is an SPROC program and hence is interpreted in a usual manner (§3.2). The outcome of each section is bundled up and constitutes the outcome of the original $\text{SPROC}^{\mathtt{dt}}$ program. In §4.2 we formalize this idea in the NSA terms of §2.2.

Definition 4.2 (Section $e|_i$ of $\text{SPROC}^{\mathtt{dt}}$ expressions)**.** Let p be an $\text{SPROC}^{\mathtt{dt}}$ expression. For each $i\in\mathbb{N}$, its *i-th section* $p|_i$ is the SPROC expression obtained from p, by replacing 1) the stream expression dt with $\frac{1}{i+1}$; 2) the operator $\mathtt{fby}^{\frac{r}{\mathtt{dt}}}$ with $\mathtt{fby}^{\lceil r(i+1)\rceil}$.

4.2 $\text{SPROC}^{\mathtt{dt}}$: Semantics

We repeat the development of the semantics of SPROC—replacing streams with hyperstreams, via *-transform—and interpret $\text{SPROC}^{\mathtt{dt}}$. Here we rely on domain theory formulated in an NSA setting. Its details are found in Appendix B (part of which already appeared in [3]).

Definition 4.3 (Variable/node environment for $\text{SPROC}^{\mathtt{dt}}$)**.** The set of *(stream) variable environments* for $\text{SPROC}^{\mathtt{dt}}$ is the *-transform ${}^*\mathbf{SVarEnv}$ of $\mathbf{SVarEnv}$ for SPROC. Recall that $\mathbf{SVarEnv}=(\mathbf{SVar}\to\mathbb{C}^\infty)$; by Lem. 2.12, a variable environment for $\text{SPROC}^{\mathtt{dt}}$ is precisely an internal function $J:{}^*\mathbf{SVar}\to{}^*(\mathbb{C}^\infty)$.

Similarly, the set of *node environments* for $\text{SPROC}^{\mathtt{dt}}$ is the *-transform ${}^*\mathbf{NdEnv}$ of that for SPROC.

We shall first define the denotation $[\![e]\!]_{J,K}\in{}^*(\mathbb{C}^\infty)$ of a hyperstream expression $e\in\mathbf{SExp}_{\mathbb{C}}$ in $\text{SPROC}^{\mathtt{dt}}$, under $J\in{}^*\mathbf{SVarEnv}$ and $K\in{}^*\mathbf{NdEnv}$. This is done sectionwise; we proceed exploiting the NSA machinery in §2.2, finally leading to Def. 4.4.

Given $e\in\mathbf{SExp}_{\mathbb{C}}$ in $\text{SPROC}^{\mathtt{dt}}$, each section $e|_i$ (Def. 4.2) is an SPROC expression. Its denotation (Def. 3.7) yields a function

$$[\![e|_i]\!]\ :\ \mathbf{SVarEnv}\times\mathbf{NdEnv}\longrightarrow_{\mathrm{ct}}\mathbb{C}^\infty\ ;$$

its continuity is proved in Lem. A.3. We collect $[\![e|_i]\!]$ for each i; this results, using Lem. 2.13, in the following function.

$$\begin{aligned} M\big[\big([\![e|_i]\!]\big)_{i\in\mathbb{N}}\big] &\in {}^*(\mathbf{SVarEnv}\times\mathbf{NdEnv}\to_{\mathrm{ct}}\mathbb{C}^\infty)\\ &\overset{\text{Lem. B.6 \& 2.10}}{=} ({}^*\mathbf{SVarEnv}\times{}^*\mathbf{NdEnv}\to_{{}^*\mathrm{ct}}{}^*(\mathbb{C}^\infty)) \end{aligned} \tag{22}$$

The last denotes the space of **-continuous functions* (Def. B.5), whose details can safely be skipped for the moment.

Definition 4.4 ($[\![e]\!]$, $[\![b]\!]$)**.** The *denotation* $[\![e]\!]$, of a $\mathbb{C}$-hyperstream expression $e\in\mathbf{SExp}_{\mathbb{C}}$ in $\text{SPROC}^{\mathtt{dt}}$, is defined as follows using (22).

$$[\![e]\!] := M\big[([\![e|_i]\!])_{i\in\mathbb{N}}\big] \;:\; {}^*\mathbf{SVarEnv}\times{}^*\mathbf{NdEnv}\to_{{}^*\mathrm{ct}} {}^*(\mathbb{C}^\infty)$$

The denotation $[\![b]\!] : {}^*\mathbf{SVarEnv}\times{}^*\mathbf{NdEnv}\to_{{}^*\mathrm{ct}} {}^*(\mathbb{B}^\infty)$ of $b\in\mathbf{SExp}_{\mathbb{B}}$ in SPROC$^{\mathrm{dt}}$ is defined in the same manner.

We now interpret nodes and programs in SPROC$^{\mathrm{dt}}$. There are two equivalent ways to do so; here we present the "sectionwise" definition that is similar to Def. 4.4. This is more convenient for the later use in §4.3; see Rem. 4.6 for the other definition.

Definition 4.5 ($[\![\mathsf{nd}]\!]$, $[\![\mathsf{pg}]\!]$)**.** Given a node nd in SPROC$^{\mathrm{dt}}$ of arity (m,n), its denotation $[\![\mathsf{nd}]\!]$ is defined by

$$[\![\mathsf{nd}]\!] := M\big[\,([\![\mathsf{nd}|_i]\!])_{i\in\mathbb{N}}\,\big] \;: \; {}^*\mathbf{NdEnv}\to_{{}^*\mathrm{ct}}\big({}^*(\mathbb{C}^\infty)^m\to_{{}^*\mathrm{ct}}{}^*(\mathbb{C}^\infty)^n\big)\;,$$

where $[\![\mathsf{nd}|_i]\!] : \mathbf{NdEnv}\to_{\mathrm{ct}} ((\mathbb{C}^\infty)^m\to_{\mathrm{ct}}(\mathbb{C}^\infty)^n)$ is as defined in Def. 3.8 (its continuity is proved in Lem. A.6).

For a program pg in SPROC$^{\mathrm{dt}}$ of arity (m,n), its denotation $[\![\mathsf{pg}]\!]$ is defined similarly by

$$[\![\mathsf{pg}]\!] := M\big[\,([\![\mathsf{pg}|_i]\!])_{i\in\mathbb{N}}\,\big] \;:\; {}^*(\mathbb{C}^\infty)^{m_{\mathsf{Main}}}\to_{{}^*\mathrm{ct}}{}^*(\mathbb{C}^\infty)^{n_{\mathsf{Main}}}\;.$$

Remark 4.6. A drawback of the sectionwise definition of $[\![\mathsf{nd}]\!]$ (Def. 4.5) is that the relationship between $[\![\mathsf{nd}]\!]$ and $[\![e]\!]$ (for e occurring in nd) is not visible at all. Conceptually this is unnatural.

In fact, we can define $[\![\mathsf{nd}]\!]$ directly from $[\![e]\!]$—like we did in §3.2—by solving a "hyperdomain equation" in the hyperdomain ${}^*\mathbf{SVarEnv}$. For the latter we use the technique presented in [3]; see Appendix B, especially Lem. B.7. The two definitions indeed coincide; see Appendix C for details.

4.3 SPROC$^{\mathrm{dt}}$: Type System for Safety Guarantee

We introduce a type system for SPROC$^{\mathrm{dt}}$ as a "*-transform" of that for SPROC. It might be hard at this stage to make sense of a hyperstream s satisfying a type τ; it will be used in our main theorem (Thm. 5.12).

4.3.1 SPROC$^{\mathrm{dt}}$: Type Syntax

$$\begin{array}{rcl}
\mathbf{AExp}\ni a & ::= & v \mid c \mid a_1+a_2 \mid a_1\times a_2 \mid a_1\,\hat{}\,a_2 \mid \lceil a_1\rceil \mid \\
 & & \mathtt{dt} \mid \frac{1}{\mathtt{dt}} \quad \text{where } v\in\mathbf{Var} \text{ and } c\in\mathbb{C}\\
\mathbf{Fml}\ni P & ::= & \mathtt{true} \mid \mathtt{false} \mid P_1\wedge P_2 \mid P_1\vee P_2 \mid \neg P \mid \\
 & & a_1=a_2 \mid \mathtt{isReal}(a) \mid a_1<a_2 \mid a_1\le a_2 \mid \\
 & & \forall v\in{}^*\mathbb{N}.\,P \mid \forall v\in{}^*\mathbb{C}.\,P \\
 & & \quad \text{where } v\in\mathbf{Var} \text{ and } a, a_i\in\mathbf{AExp}\\
\mathbf{SType}_{\mathbb{C}}\ni\tau & ::= & \prod_{v\in{}^*\mathbb{N}}\{u\in{}^*\mathbb{C}\mid P\} \quad \text{where } u,v\in\mathbf{Var},\\
 & & P\in\mathbf{Fml} \text{ and } \mathrm{FV}(P)\subseteq\{u,v\}\\
\mathbf{SType}_{\mathbb{B}}\ni\beta & ::= & \prod_{v\in{}^*\mathbb{N}} P \quad \text{where } v\in\mathbf{Var},\\
 & & P\in\mathbf{Fml} \text{ and } \mathrm{FV}(P)\subseteq\{v\}\\
\mathbf{NdType}_{m,n}\ni\nu & ::= & (\tau_1,\dots,\tau_m)\to(\tau'_1,\dots,\tau'_n)\\
 & & \quad \text{where } \tau_i,\tau'_i\in\mathbf{SType}_{\mathbb{C}}
\end{array}$$

Table 5. Type Syntax for SPROC$^{\mathrm{dt}}$

Definition 4.7 (Types for SPROC$^{\mathrm{dt}}$)**.** The syntax of the SPROC$^{\mathrm{dt}}$ type system is in Table 5. It is almost the same as that for SPROC (Table 3). The differences are: 1) we have $\mathtt{dt}\in\mathbf{AExp}$ that represents an infinitesimal sampling interval; 2) quantifiers in $\mathbf{Fml}$ and stream types are taken over hypernumbers ${}^*\mathbb{N}$, ${}^*\mathbb{C}$, instead of standard numbers.

We define sections of type expressions. This is like Def. 4.2.

Definition 4.8 (Section of type expressions)**.** The *i-th section* $p|_i$ of an SPROC$^{\mathrm{dt}}$ type expression p is obtained from p by: 1) replacing dt with $\frac{1}{i+1}$; 2) replacing $\frac{1}{\mathtt{dt}}$ with $i+1$; 3) replacing hyperquantifiers $\forall v\in{}^*\mathbb{D}$ (where $\mathbb{D}\in\{\mathbb{N},\mathbb{C}\}$) with standard quantifiers $\forall v\in\mathbb{D}$; and 4) replacing hyperquantifiers $v\in{}^*\mathbb{N}$ and $u\in{}^*\mathbb{C}$ in the stream types $\prod_{v\in{}^*\mathbb{N}}\{u\in{}^*\mathbb{C}\mid P\}$ and $\prod_{v\in{}^*\mathbb{N}}P$ by the corresponding standard quantifiers. A section $p|_i$ is obviously an SPROC type expression.

4.3.2 SPROC$^{\mathrm{dt}}$: Type Semantics

Definition 4.9 (Valuation for SPROC$^{\mathrm{dt}}$)**.** The set of *valuations* for SPROC$^{\mathrm{dt}}$ is ${}^*\mathbf{Val}$, the *-transform of the set $\mathbf{Val}=(\mathbf{Var}\to\mathbb{C})\cup\{\bot\}$ in Def. 3.15. By 2.10, a valuation for SPROC$^{\mathrm{dt}}$ is either an internal function $L:{}^*\mathbf{Var}\to{}^*\mathbb{C}$, or $L=\bot$.

The function update $L[\overrightarrow{u_i}\mapsto\overrightarrow{c_i}]$, with $L\in{}^*\mathbf{Val}$, $u_i\in\mathbf{Var}$ and $c_i\in{}^*\mathbb{C}\cup\{\bot\}$, is the *-transform of the corresponding operation in SPROC (Def. 3.15). Namely, the latter induces a function (by (21))

$$\begin{array}{l}\Theta:\mathbf{Val}\times\big(\mathbf{Var}\times(\mathbb{C}\cup\{\bot\})\big)^m\to\mathbf{Val},\\ \qquad (L',\overrightarrow{(u',c')})\mapsto L'[\overrightarrow{u'}\mapsto\overrightarrow{c'}].\end{array} \tag{23}$$

Its *-transform under ${}^*(\_)$ in (9) is an internal function ${}^*\Theta : {}^*\mathbf{Val}\times\big({}^*\mathbf{Var}\times({}^*\mathbb{C}\cup\{\bot\})\big)^m\to{}^*\mathbf{Val}$. We precompose the injection $\mathbf{Var}\hookrightarrow{}^*\mathbf{Var}$ from Lem. 2.10.1, and obtain

$$\Theta' \;:\; {}^*\mathbf{Val}\times\big(\mathbf{Var}\times({}^*\mathbb{C}\cup\{\bot\})\big)^m\longrightarrow{}^*\mathbf{Val}\;.$$

We define the function update by $L[\vec{u}\mapsto\vec{c}] := \Theta'(L,\overrightarrow{(u,c)})$.

Definition 4.10 (Semantics of $\mathbf{AExp}$, $\mathbf{Fml}$)**.** The denotation $[\![a]\!]_L\in{}^*\mathbb{C}\cup\{\bot\}$ of $a\in\mathbf{AExp}$ under a valuation $L\in{}^*\mathbf{Val}$ is defined as follows. It is $\bot$ when $L=\bot$; otherwise

$$\begin{array}{rcl}
[\![v]\!]_L & := & L({}^*v)\;, \qquad [\![c]\!]_L := {}^*c\;,\\
[\![\mathtt{dt}]\!]_L & := & M\big[(\tfrac{1}{i+1})_{i\in I}\big]=[(1,\tfrac12,\tfrac13,\dots)]\;,\\
[\![a_1\,\mathtt{aop}\,a_2]\!]_L & := & [\![a_1]\!]_L\;{}^*\mathtt{aop}\;[\![a_2]\!]_L \quad \text{where } \mathtt{aop}\in\{+,\times,\hat{}\}.
\end{array}$$

In the first line, ${}^*(\_):\mathbf{Var}\to{}^*\mathbf{Var}$ and ${}^*(\_):\mathbb{C}\to{}^*\mathbb{C}$ are from Lem. 2.10.1. In the second line recall that $I=\mathbb{N}$ (Rem. 2.3); thus $[\![\mathtt{dt}]\!]_L$ is in fact the infinitesimal number ω^{-1} exhibited in §2.1. In the last line, ${}^*\mathtt{aop}:({}^*\mathbb{C})^2\to{}^*\mathbb{C}$ is the *-transform of aop.

The *satisfaction relation* $L\models P$ between $L\in{}^*\mathbf{Val}$ and $P\in\mathbf{Fml}$ is defined in the usual manner. For example,

$$\begin{array}{l} L\models a_1<a_2 \quad\stackrel{\text{def.}}{\iff}\\ \qquad L=\bot \text{ or } \big([\![a_1]\!]_L,[\![a_2]\!]_L\in{}^*\mathbb{R}\;\wedge\;[\![a_1]\!]_L\,{}^*\!\!<_\bot[\![a_2]\!]_L\big)\;.\end{array}$$

A formula P is *valid* (written $\models P$) if $L\models P$ for any $L\in{}^*\mathbf{Val}$.

We shall characterize this semantics in a sectionwise manner, so that we can later apply Łoś' theorem (Lem. 2.13). For each $a\in\mathbf{AExp}$ in SPROC$^{\mathrm{dt}}$, its section $a|_i$ determines by Def. 3.16 a function $[\![a|_i]\!]:\mathbf{Val}\to\mathbb{C}\cup\{\bot\}$. Thus by Lem. 2.13 we have

$$M\big[([\![a|_i]\!])_{i\in I}\big]\;\in{}^*(\mathbf{Val}\to\mathbb{C}\cup\{\bot\})\stackrel{\text{Lem. 2.10}}{\subseteq}({}^*\mathbf{Val}\to{}^*\mathbb{C}\cup\{\bot\})\;. \tag{24}$$

Similarly, a formula $P\in\mathbf{Fml}$ for SPROC$^{\mathrm{dt}}$ determine

$$\begin{array}{ll} M\big[(\,\_\models P|_i\,)_{i\in I}\big] & \in{}^*(\mathbf{Val}\to\mathbb{B})\subseteq({}^*\mathbf{Val}\to\mathbb{B})\;,\\ M\big[(\models P|_i\,)_{i\in I}\big] & \in{}^*\mathbb{B}\stackrel{\cong}{\to}\mathbb{B}.\end{array} \tag{25}$$

Lemma 4.11. *Between Def. 4.10 and (24–25), the following hold.*

$$\begin{array}{l}[\![a]\!]_L \;=\;\big(M[([\![a|_i]\!])_{i\in I}]\big)(L)\\ L\models P \iff \big(M\big[(\,\_\models P|_i\,)_{i\in I}\big]\big)(L)=\mathtt{tt}\\ \models P \iff M\big[(\models P|_i\,)_{i\in I}\big]=\mathtt{tt}\end{array}$$

□

The next definition is parallel to Def. 3.17.

Definition 4.12. The *satisfaction relation* $\models$ between a hyperstream

$$s\in{}^*(\mathbb{C}^\infty)\subseteq({}^*\mathbb{N}\to{}^*\mathbb{C}\cup\{\bot\}) \quad (\text{where }\subseteq\text{ is due to Lem. B.3})$$

and a $\mathbb{C}$-hyperstream type $\tau\equiv\prod_{v\in{}^*\mathbb{N}}\{u\in{}^*\mathbb{C}\mid P\}$ is defined by:

$$\begin{array}{l} s\models\prod_{v\in{}^*\mathbb{N}}\{u\in{}^*\mathbb{C}\mid P\} \quad\stackrel{\text{def.}}{\iff}\\ \qquad L[v\mapsto n, u\mapsto s(n)]\models P \text{ for each } n\in{}^*\mathbb{N} \text{ and } L\in{}^*\mathbf{Val}.\end{array}$$

In the setting of the previous definition, each section $\tau|_i \equiv \prod_{v\in\mathbb{N}}\{u \in \mathbb{C} \mid P|_i\}$ determines a function (by Def. 3.17)

$$(\_ \models \textstyle\prod_{v\in\mathbb{N}}\{u \in \mathbb{C} \mid P|_i\}) \;:\; \mathbb{C}^\infty \longrightarrow \mathbb{B} \;;$$

therefore by Lem. 2.13 we obtain

$$M[(\_ \models \textstyle\prod_{v\in\mathbb{N}}\{u \in \mathbb{C} \mid P|_i\})_{i\in I}] \;:\; {}^*(\mathbb{C}^\infty) \longrightarrow \mathbb{B} \;. \tag{26}$$

Lemma 4.13. *Between Def. 4.12 and (26), the following holds.*

$$s \models \tau \iff \Big(M[(\_ \models \textstyle\prod_{v\in\mathbb{N}}\{u \in \mathbb{C} \mid P|_i\})_{i\in I}]\Big)(s) = \mathtt{tt}$$

□

Similarly, for $\mathbb{B}$-hyperstream types and node types, the satisfaction relations $\models$ are defined much like in Def. 3.17. See Appendix D.6. They have the following sectionwise characterizations, too.

Lemma 4.14.

$$t \models \textstyle\prod_{v\in{}^*\mathbb{N}} P \iff \Big(M[(\_ \models \prod_{v\in\mathbb{N}} P|_i\,)_{i\in I}]\Big)(t) = \mathtt{tt} \;;$$

$$g \models (\vec{\tau}) \to (\vec{\tau'}) \iff \Big(M[(\_ \models (\overrightarrow{\tau|_i}) \to (\overrightarrow{\tau'|_i}))_{i\in I}]\Big)(g) = \mathtt{tt} \quad □$$

4.3.3 SPROC$^{\mathsf{dt}}$: Type Derivation

Typing rules in SPROC$^{\mathsf{dt}}$ are almost the same as in SPROC. In particular the use of hypernumbers is transparent; this reflects the NSA idea that standard numbers and hypernumbers are logically the same.

Definition 4.15 (Type environment)**.** A *stream type environment* and a *node type environment* for SPROC$^{\mathsf{dt}}$ are defined in the same way as in SPROC (Def. 3.18): the former is a finite subset $\Gamma = \{x_i : \tau_i\} \subseteq \mathbf{Var} \times \mathbf{SType}_{\mathbb{C}}$. We denote the sets of stream and node type environments by $\mathbf{STEnv}$ and $\mathbf{NdTEnv}$, respectively.

Definition 4.16 (Type derivation)**.** The type judgments and the typing rules for SPROC$^{\mathsf{dt}}$ are the same as for SPROC (Table 4), except for:

- they are *-transformed, that is, the quantifiers ($\prod_{v\in\mathbb{N}}\{u \in \mathbb{C} \mid \_\}$, $\forall x \in \mathbb{C}$, etc.) are replaced by the corresponding hyperquantifiers ($\prod_{v\in{}^*\mathbb{N}}\{u \in {}^*\mathbb{C} \mid \_\}$, $\forall x \in {}^*\mathbb{C}$, etc.);
- we have the following two additional rules. (FBY$^{\frac{r}{\mathsf{dt}}}$) is similar to (FBY^j); there $\lceil\frac{r}{\mathsf{dt}}\rceil$ is short for $\lceil r \times \frac{1}{\mathsf{dt}}\rceil$.

$$\frac{}{\Delta;\Gamma \vdash \mathsf{dt} : \prod_{v\in\mathbb{N}}\{u \in \mathbb{C} \mid u = \mathsf{dt}\}}\;(\mathsf{dt})$$

$$\frac{\begin{array}{c}\Delta;\Gamma \vdash e_1 : \prod_v\{u \mid P_1\} \qquad \Delta;\Gamma \vdash e_2 : \prod_v\{u \mid P_2\} \\ \models \forall v \in {}^*\mathbb{N}.\forall u \in {}^*\mathbb{C}.\big((v < \frac{r}{\mathsf{dt}} \wedge P_1 \;\Rightarrow P)\wedge \\ (v \geq \frac{r}{\mathsf{dt}} \wedge P_2[(v - \lceil\frac{r}{\mathsf{dt}}\rceil)/v] \;\Rightarrow P)\big)\end{array}}{\Delta;\Gamma \vdash e_1\; \mathtt{fby}^{\frac{r}{\mathsf{dt}}}\; e_2 : \prod_{v\in{}^*\mathbb{N}}\{u \in {}^*\mathbb{C} \mid P\}}\;(\mathrm{FBY}^{\frac{r}{\mathsf{dt}}})$$

4.3.4 SPROC$^{\mathsf{dt}}$: Type Soundness

Definition 4.17. The *satisfaction relation* $J \models \Gamma$ between $J \in {}^*\mathbf{SVarEnv}$ (Def. 4.3) and $\Gamma \in \mathbf{STEnv}$ (Def. 4.15) is

$$J \models \Gamma \quad \stackrel{\text{def.}}{\iff} \quad J({}^*x_i) \models \Gamma(x_i) \text{ for each } i \in [1, m],$$

where ${}^*x_i$ is the image under ${}^*(\_) : \mathbf{Var} \to {}^*\mathbf{Var}$ (Lem. 2.10.1). The *satisfaction relation* $K \models \Delta$ between $K \in {}^*\mathbf{NdEnv}$ and $\Delta \in \mathbf{NdTEnv}$ is defined in the same way.

Definition 4.18 (Validity of type judgments)**.** We say a type judgment $\Delta;\Gamma \vdash e : \tau$ is *valid*, and write $\models \;\; \Delta;\Gamma \vdash e : \tau$, if for any $J \in {}^*\mathbf{SVarEnv}$ and $K \in {}^*\mathbf{NdEnv}$, $J \models \Gamma$ and $K \models \Delta$ imply $[\![e]\!]_{J,K} \models \tau$. The *validity* of the other three classes of type judgments is defined in the same manner.

Lemma 4.19. *Validity of a judgment is determined sectionwise:*

$$\models \Delta;\Gamma \vdash e : \tau \iff M\big[(\models \Delta|_i;\Gamma|_i \vdash e|_i : \tau|_i)_{i\in I}\big] = \mathtt{tt} \;.$$

□

Finally we come to soundness. Its proof is totally modular, exploiting the sectionwise characterizations of $\models$'s. It is notable that the content of rules does not matter, as long as they are sound for SPROC.

Theorem 4.20 (Type soundness of SPROC$^{\mathsf{dt}}$)**.** *A derivable type judgment is valid in* SPROC$^{\mathsf{dt}}$*, that is,* $\Vdash \mathcal{J}$ *implies* $\models \mathcal{J}$*.* □

5. Signals as Hyperstreams

We introduce a translation between (continuous-time) signals and hyperstreams. This enables SPROC$^{\mathsf{dt}}$ to model signals, and its type system to provide signals' safety guarantees. Such signals cannot just be *any* function $f : \mathbb{R}_{\geq 0} \to \mathbb{C}$; we introduce a certain class of functions that makes the translation work (Def. 5.1). The class is closed under common operations like integration and differentiation, too.

The basic idea of a translation between signals and hyperstreams (§5.2) is already in [3], where they establish the correctness of their translation for functions $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ that are everywhere continuous (they also hint an extension to piecewise continuity). Since we aim at hybrid applications, our class of functions (Def. 5.1) is broader and contains some Zeno examples such as a bouncing ball.

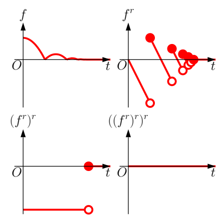
In §5.1–5.2, for simplicity, we prove results for the $\mathbb{R}$-valued signals and streams. Extension to $\mathbb{C}$-valued ones is straightforward—we can separate real and imaginary parts and identify $\mathbb{C}$ with $\mathbb{R}^2$.

5.1 Signals

Functions that are right continuous with left limits everywhere play an important role in the theory of stochastic processes. They are called *càdlàg*; our class of (continuous enough) *signal* is based on this.

Definition 5.1 (Signal)**.** A function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ is *of class Càdlàg*^{n} if: 1) it is right differentiable; 2) it has left limits $\lim_{t\to t_0-0} f(t)$ for each $t_0 \in \mathbb{R}_{>0}$; and 3) its right derivative f^r is of class *Càdlàg*$^{n-1}$. A function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ is *of class Càdlàg*^{0} if it is right continuous and has left limits everywhere. A function f is *of class Càdlàg*$^\infty$ if it is of class *Càdlàg*^{n} for all $n \in \mathbb{N}$.

A function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ is said to be a *signal* if it is of class *Càdlàg*$^\infty$ and, for any $t \in \mathbb{R}_{\geq 0}$, there is $\varepsilon > 0$ such that f is of class C^∞ in the interval $(t, t+\varepsilon)$. **Signals** denotes the set of signals.



Many common hybrid dynamics are indeed signals in our sense; but not all of them. A bouncing ball—a first example of the *Zeno behavior*—is modeled as a signal (on the right). However, if we reverse time and flip horizontally, the resulting is not a signal: the halting point t_0 has no interval $(t_0, t_0+\varepsilon)$ in which the function is of class C^∞.

Another nonexample arises from the *compare-to-constant* operation in Simulink. The function

$$f(t) = \quad e^{-\frac{1}{t}} \sin \tfrac{1}{t} \quad (\text{if } t > 0) \;; \qquad 0 \quad (\text{if } t \leq 0)$$

oscillates around $t = 0$ very fast but very small (due to the factor $e^{-\frac{1}{t}}$); it is a signal. However, comparison with 0 results in a nonsignal—it is clearly not right continuous at $t = 0$.

Our notion of signal still has reasonable closure properties.

Lemma 5.2. *A signal $f \in$ **Signals** is right differentiable and Riemann integrable, resulting again in signals.* □

Remark 5.3. The notion of càdlàg function is used mostly in the context of stochastic systems. This is also the case in the hybrid system literature [8, 22]. Our use of the notion suggests it might also be related to the question of *samplability* (see e.g. [34]), though the details are yet to be worked out.

5.2 Signals as Hyperstreams

We define the *(hyperstream) sampling map* Smp and the *smoothing map* Smth, and show that they form faithful translation of signals into hyperstreams, that is roughly, $\mathsf{Smth}(\mathsf{Smp}(f)) = f$. Recall that in §5.1–5.2 we are restricting to $\mathbb{R}$-valued hyperstreams.

$$\begin{tikzcd} \mathbf{Signals} \arrow[r, "\mathsf{Smp}"'] \arrow[d, hook] & {}^*(\mathbb{R}^\infty) \arrow[dl, "\mathsf{Smth}"] \\ (\mathbb{R}_{\geq 0} \to {}^*\mathbb{R} \cup \{\bot\}) & \end{tikzcd}$$

Definition 5.4 (Smp)**.** $\mathsf{Smp} : \mathbf{Signals} \to {}^*(\mathbb{R}^\infty)$ is defined by

$$\mathsf{Smp}(f) \;:=\; M\Big[\Big((f(\tfrac{j}{i+1}))_{j\in\mathbb{N}} \Big)_{i \in I}\Big] \;.$$

This is exactly the hyperstream sampling (2) put in the NSA terms. Here $(f(\frac{j}{i+1}))_{j\in\mathbb{N}}$ is in $\mathbb{R}^\infty$, for each $i \in I = \mathbb{N}$; hence by Lem. 2.13 we have $M\big[\big((f(\frac{j}{i+1}))_{j\in\mathbb{N}}\big)_{i\in I}\big]$ belong to ${}^*(\mathbb{R}^\infty)$.

The converse *smoothing* operation (3) need not yield a signal, or even a function $\mathbb{R}_{\geq 0} \to \mathbb{R}$. The stream in (3) need not converge; that is, in the NSA terms, the hyperreal $\big[\big(f(\lceil t \rceil),\, f(\frac{\lceil 2t \rceil}{2}),\, \dots\big)\big]$ can be an infinite number. This results in the extended output type $\mathbb{R}_{\geq 0} \to {}^*\mathbb{R} \cup \{\bot\}$ in the following definition of Smth.

Definition 5.5 (Smth)**.** The mapping $\mathsf{Smth} : {}^*(\mathbb{R}^\infty) \to (\mathbb{R}_{\geq 0} \to {}^*\mathbb{R} \cup \{\bot\})$ is defined as follows. Let $h \in {}^*(\mathbb{R}^\infty)$ and $t \in \mathbb{R}_{\geq 0}$; the latter induces a function $(\lceil (i+1)t \rceil)_{i \in I}$ from $I = \mathbb{N}$ to $\mathbb{N}$. Lem. 2.13 yields $M\big[\,(\lceil (i+1)t \rceil)_{i\in I}\,\big]$ as an element of ${}^*\mathbb{N}$; this is fed to $h \in {}^*(\mathbb{R}^\infty) \subseteq ({}^*\mathbb{N} \to {}^*\mathbb{R} \cup \{\bot\})$ (Lem. B.3) and we define

$$\mathsf{Smth}(h)(t) \;:=\; h\Big(M\big[\,(\lceil (i+1)t \rceil)_{i \in I}\,\big] \Big) \;.$$

"$\mathsf{Smth} \circ \mathsf{Smp} = \mathrm{id}$" is put precise using shadow (Def. 2.2).

Theorem 5.6. *For each $f \in \mathbf{Signals}$ and $t \in \mathbb{R}_{\geq 0}$, the hyperreal $\mathsf{Smth}(\mathsf{Smp}(f))(t)$ is limited and $\mathrm{sh}\big(\mathsf{Smth}(\mathsf{Smp}(f))(t)\big) = f(t)$.* □

5.3 Modeling Signals in SPROC$^{\mathrm{dt}}$

From this point on we are back in the $\mathbb{C}$-valued setting. We rely on the definitions and results in §5.1–5.2 extended from $\mathbb{R}$ to $\mathbb{C}$.

Soundness of signal verification via SPROC$^{\mathrm{dt}}$ relies on the correct modeling of a signal f as an SPROC$^{\mathrm{dt}}$ program pg_f (cf. the usage scenario in §1). Its extensive treatment—especially the translation of ODEs and Simulink diagrams into SPROC$^{\mathrm{dt}}$ programs—will be presented in another venue. Here we present some basic results.

Definition 5.7 (SPROC$^{\mathrm{dt}}$ model)**.** Let $f \in \mathbf{Signals}$. A hyperstream expression $e \in \mathbf{SExp}_{\mathbb{C}}$ in SPROC$^{\mathrm{dt}}$ is said to be a *model* of f under J, K, if $\mathrm{sh}\big(\mathsf{Smth}(\llbracket e \rrbracket_{J,K})(t)\big) = f(t)$ for all $t \in \mathbb{R}_{\geq 0}$. It is similarly defined for an SPROC$^{\mathrm{dt}}$ program pg to be a *model* of f.

Proposition 5.8. *Let e_1, e_2 be models of signals f_1, f_2 under J, K.*

1. *For each $c \in \mathbb{C}$, the constant symbol $c \in \mathbf{SExp}_{\mathbb{C}}$ is a model of the constant signal $\overline{c}(t) = c$.*
2. *$(e_1 \;\mathtt{aop}\; e_2)$ is a model of the signal $(f_1 \;\mathtt{aop}\; f_2)$ (computed pointwise) under J, K, for $\mathtt{aop} \in \{+, \times, \hat{}\,\}$.*
3. *$(e_1 \;\mathtt{fby}^{\frac{r}{\mathrm{dt}}}\; e_2)$ is a model, under J, K, of the signal $(f_1 \;\mathtt{fby}^{r\text{ sec.}}\; f_2)$. The latter is defined below; it is easily seen to be a signal.*

$$\big(\, f_1 \;\mathtt{fby}^{r\text{ sec.}}\; f_2 \,\big)(t) := \begin{cases} f_1(t) & \text{if } t < r, \\ f_2(t-r) & \text{if } t \geq r. \end{cases}$$ □

As to Lem. 5.2, we also have SPROC$^{\mathrm{dt}}$ programs for right differentiation and Riemann integration. We leave them to another venue.

In Example 1.1 we have used an SPROC$^{\mathrm{dt}}$ model (6) of the sine curve, where the latter is defined using an ODE. Its (intuitively obvious) correctness can be proved by showing that $\mathsf{Smth}(\llbracket \mathsf{pg}_{\mathsf{Sine}} \rrbracket)$ is a solution of the ODE defining the sine curve.

5.4 Safety Guarantee for Signals

In translating safety guarantees from SPROC$^{\mathrm{dt}}$ to signals, the property $\tau \equiv \prod_v \{u \mid P\}$ cannot be just anything—after all, Smp samples only countably many $t \in \mathbb{R}$. A sufficient condition is given by τ's being *(topologically) closed*. It means that the set $\{(u, v) \mid\, \models P\}$ is closed in $\mathbb{C}^2$. The type syntax for signals is restricted accordingly; in particular, adding $<$, $\neg$ or $\exists$ makes topological closedness fail.

Definition 5.9 (Type Syntax for Signals)**.**

$$\begin{array}{rcl} \mathbf{AExp} \ni a & ::= & v \mid c \mid a_1 \;\mathtt{aop}\; a_2 \\ & & \quad \text{where } v \in \mathbf{Var},\, c \in \mathbb{C},\, \mathtt{aop} \in \{+, \times, \hat{}\,\} \\ \mathbf{Fml} \ni P & ::= & \mathtt{true} \mid \mathtt{false} \mid P_1 \vee P_2 \mid P_1 \wedge P_2 \mid a_1 = a_2 \mid \\ & & \mathtt{isReal}(a) \mid a_1 \leq a_2 \mid \forall v \in \mathbb{C}.\, P \\ & & \quad \text{where } v \in \mathbf{Var},\, a, a_i \in \mathbf{AExp} \\ \mathbf{SgType} \ni \tau & ::= & \prod_{w \in \mathbb{R}_{\geq 0}} \{u \in \mathbb{C} \mid P\} \quad \text{where } u, w \in \mathbf{Var}, \\ & & \quad P \in \mathbf{Fml} \text{ and } \mathrm{FV}(P) \subseteq \{u, w\} \\ \mathbf{SgPrType}_{m,n} \ni \nu & ::= & (\tau_1, \dots, \tau_m) \to (\tau'_1, \dots, \tau'_n) \\ & & \quad \text{where } \tau_i, \tau'_i \in \mathbf{SgType} \end{array}$$

The set $\mathbf{SgType}$ is that of *signal types*.

Definition 5.10 (Semantics of Signal Types)**.** A *valuation* L is the same as in Def. 3.15; $L \models P$ with $P \in \mathbf{Fml}$ is defined as usual, too.

Between $f \in \mathbf{Signals}$ and $\tau \in \mathbf{SgType}$, $f \models \tau$ is defined by:

$$\begin{array}{l} f \models \prod_{w \in \mathbb{R}_{\geq 0}} \{u \in \mathbb{C} \mid P\} \;\stackrel{\text{def.}}{\Longleftrightarrow} \\ \quad \forall t \in \mathbb{R}_{\geq 0}.\, \forall L \in \mathbf{Val}.\; L[w \mapsto t, u \mapsto f(t)] \models P \;\;. \end{array}$$

As in Def. 3.17, L in the above is vacuous since $\mathrm{FV}(P) \subseteq \{u, w\}$.

Definition 5.11 (Translation p^{HS} of types)**.** To each type expression p for signals (Def. 5.9), we assign an SPROC$^{\mathrm{dt}}$ type expression p^{HS}. It is defined by replacing: 1) the signal type $\prod_{w \in \mathbb{R}_{\geq 0}} \{u \in \mathbb{C} \mid P\}$ with the hyperstream type $\prod_{v \in {}^*\mathbb{N}} \{u \in \mathbb{C} \mid P^{\mathrm{HS}}[(v \times \mathtt{dt})/w]\}$; 2) quantifiers $\forall v \in \mathbb{C}$ with hyperquantifiers $\forall v \in {}^*\mathbb{C}$.

The idea of p^{HS} is to represent time w in P by the step number v multiplied by the sampling interval $\mathtt{dt}$.

Theorem 5.12 (Soundness)**.** *Let $f \in \mathbf{Signals}$ be a signal, and $\tau \equiv \prod_{w \in \mathbb{R}_{\geq 0}} \{u \in \mathbb{C} \mid P\} \in \mathbf{SgType}$ be a signal type. Assume further that an* SPROC$^{\mathrm{dt}}$ *program* pg *is a model of f (Def. 5.7). Then:*

$$\Vdash \vdash \mathsf{pg} : () \to (\tau^{\mathrm{HS}}) \qquad \Longrightarrow \qquad f \models \tau \;.$$ □

We expect the opposite direction $\Leftarrow$ of the theorem to fail in general—the left-hand side guarantees safety also for a time t that is infinitely large. We also note that, while the signal type syntax (Def. 5.9) is more restricted than that of SPROC$^{\mathrm{dt}}$ (§4.3), in deriving the left-hand side of the theorem one can safely use the full SPROC$^{\mathrm{dt}}$ type system.

6. An Example: The Sine Curve

We verify the range of the sine curve. Specifically, we show that $\mathsf{pg}_{\mathsf{Sine}}$ in Example 1.1 satisfies, for any real constants $t_0 > 0$ and $\varepsilon > 0$,

$$\llbracket \mathsf{pg}_{\mathsf{Sine}} \rrbracket \;\models\; \prod_{w \in \mathbb{R}_{\geq 0}} \{u \in \mathbb{C} \mid t_0 \leq w \vee u \leq 1 + \varepsilon\} \;. \tag{27}$$

Our intuition of the formula in (27) is $w < t_0 \Rightarrow u \leq 1 + \varepsilon$; due to the restricted syntax of signal types (Def. 5.9) it is written as in (27). By Thm. 5.12, it suffices to derive the following type judgment using the typing rules of SPROC$^{\mathtt{dt}}$.

$$\begin{array}{l} \vdash \mathsf{pg}_{\mathsf{Sine}} : () \to (\tau_{\mathsf{goal}}) \ \text{, where} \\ \tau_{\mathsf{goal}} \ :\equiv \ \prod_{v \in {}^*\mathbb{N}} \{ u \in {}^*\mathbb{C} \mid t_0 \leq v \times \mathtt{dt} \vee u \leq 1 + \varepsilon \} \ . \end{array} \tag{28}$$

In its derivation, the most significant step (below) uses the principle of fixed point induction to deal with the intra-node recursion (s and c) in $\mathsf{pg}_{\mathsf{Sine}}$. This step is an instance of the (NODE) rule (Table 4):

$$\frac{\begin{array}{ll} \text{(a)} & \Delta_0; \{ s : \tau_{s\text{-inv}}, c : \tau_{c\text{-inv}} \} \vdash 0 \ \mathtt{fby}^1 \ (s + c \times \mathtt{dt}) : \tau_{s\text{-inv}} \\ \text{(b)} & \Delta_0; \{ s : \tau_{s\text{-inv}}, c : \tau_{c\text{-inv}} \} \vdash 1 \ \mathtt{fby}^1 \ (c - s \times \mathtt{dt}) : \tau_{c\text{-inv}} \\ \text{(c)} & \Delta_0; \{ s : \tau_{s\text{-inv}}, c : \tau_{c\text{-inv}} \} \vdash s : \tau_{s\text{-inv}} \end{array}}{\Delta_0; \begin{Bmatrix} s : \tau_{s\text{-inv}} \\ c : \tau_{c\text{-inv}} \end{Bmatrix} \vdash \begin{bmatrix} \mathtt{node\ Sine()\ returns}\ (s) \\ \mathtt{where}\ s = 0 \ \mathtt{fby}^1 \ (s + c \times \mathtt{dt}); \\ c = 1 \ \mathtt{fby}^1 \ (c - s \times \mathtt{dt}) \end{bmatrix} : \begin{array}{c} () \to \\ (\tau_{s\text{-inv}}) \end{array}} \tag{29}$$

Here the type environment $\Gamma_{\mathsf{inv}} := \{ s : \tau_{s\text{-inv}}, c : \tau_{c\text{-inv}} \}$ plays the role of an invariant. The types are concretely as follows.

$$\begin{array}{l} \tau_{s\text{-inv}} \ \equiv \ \prod_{v \in {}^*\mathbb{N}} \{ u \in {}^*\mathbb{C} \mid u = \frac{1}{2}\mathrm{i}(1 - \mathrm{i} \cdot \mathtt{dt})^v - \frac{1}{2}\mathrm{i}(1 + \mathrm{i} \cdot \mathtt{dt})^v \} \\ \tau_{c\text{-inv}} \ \equiv \ \prod_{v \in {}^*\mathbb{N}} \{ u \in {}^*\mathbb{C} \mid u = \frac{1}{2}(1 - \mathrm{i} \cdot \mathtt{dt})^v + \frac{1}{2}(1 + \mathrm{i} \cdot \mathtt{dt})^v \} \end{array}$$

Here i is the imaginary unit. Δ_0 in (29) is $\{ \mathtt{Sine} : () \to \tau_{\mathsf{goal}} \}$.

It is straightforward to derive the assumptions (a–c) of (29). The derivation of (a) is in Appendix D.12. (b) is similar; (c) is by (SVAR).

Therefore we have derived the conclusion of (29). To it we apply the (NDCONSEQ) and (PROG) rules and derive our final goal $\vdash \mathsf{pg}_{\mathsf{Sine}} : () \to (\tau_{\mathsf{goal}})$. The former requires the side condition

$$\begin{array}{l} \models \forall v \in {}^*\mathbb{N}. \forall u \in {}^*\mathbb{C}. \left(u = \frac{1}{2}\mathrm{i}(1 - \mathrm{i} \cdot \mathtt{dt})^v - \frac{1}{2}\mathrm{i}(1 + \mathrm{i} \cdot \mathtt{dt})^v \right. \\ \qquad \left. \Rightarrow \ (t_0 \leq v \times \mathtt{dt} \vee u \leq 1 + \varepsilon) \right) \ . \end{array} \tag{30}$$

Its is proved in a discrete manner, using Lem. 2.13. See Appendix D.

As is always with the Hoare-style logics, invariant discovery is the hardest part in SPROC$^{\mathtt{dt}}$ type derivation. In this example we discovered the invariants $\tau_{s\text{-inv}}$, $\tau_{c\text{-inv}}$ by solving the recurrence relations derived from the program. This is a totally discrete business.

7. Conclusions and Future Work

Starting from a familiar framework of a stream processing language SPROC, its Kahn-style denotational semantics and a type system as a program logic, we extended it with a constant dt and obtained a framework for hyperstreams. Translation of signals into hyperstreams enables us to use deductive verification in SPROC$^{\mathtt{dt}}$ for certain safety guarantees of signals. The logical infrastructure of NSA provides the framework with a rigorous mathematical basis.

Some directions of future work are mentioned in the *related work* part of §1; here we add a couple. In this paper we have made one discretization technique (namely *discrete sampling*) "hyper" and thus exact. We are interested in use of NSA in other discretization techniques such as the Fourier transform.

Type inference for SPROC$^{\mathtt{dt}}$ is future work. Due to its character as a program logic, the situation would be much like with Hoare-style logics: even type checking (i.e. proof search) would be undecidable; and the biggest challenge would be in invariant discovery.

Acknowledgments

Thanks are due to the participants of NII Shonan Meeting *Hybrid Systems: Theory and Practice, Seriously* for useful discussions; in particular to Manuela Luminita Bujorianu for drawing our attention to càdlàg functions. We are also grateful to the anonymous referees for their useful suggestions and critical questions; and to Samuel Mimram for letting know of their work [3] (which, to our shame, we were not aware of until the last stage of the paper's preparation).

K.S. is supported by Grant-in-Aid for Research Activity Start-up No. 24800035, JSPS, and by Hakubi Project, Kyoto Univesity; H.S. & I.H. are supported by Grants-in-Aid for Young Scientists (A) No. 24680001, JSPS, and by Aihara Innovative Mathematical Modelling Project, FIRST Program, JSPS/CSTP.

References

[1] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbai, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Oxford Univ. Press, 1994.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comp. Sci.*, 138(1):3–34, 1995.

[3] R. Beauxis and S. Mimram. A non-standard semantics for Kahn networks in continuous time. In M. Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 35–50. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011. ISBN 978-3-939897-32-3.

[4] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet. Divide and recycle: types and compilation for a hybrid synchronous language. In J. Vitek and B. D. Sutter, editors, *LCTES*, pages 61–70. ACM, 2011.

[5] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet. Non-standard semantics of hybrid systems modelers. *J. Comput. Syst. Sci.*, 78(3): 877–910, 2012.

[6] S. Bliudze and D. Krob. Modelling of complex systems: Systems as dataflow machines. *Fundam. Inform.*, 91(2):251–274, 2009.

[7] O. Bouissou and A. Chapoutot. An operational semantics for Simulink's simulation engine. In R. Wilhelm, H. Falk, and W. Yi, editors, *LCTES*, pages 129–138. ACM, 2012. ISBN 978-1-4503-1212-7.

[8] M. L. Bujorianu and J. Lygeros. Theoretical foundations of stochastic hybrid systems. In *International Symposium on Mathematical Theory of Networks and Systems (MTNS 2004)*, 2004.

[9] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. Lustre: A declarative language for programming synchronous systems. In *POPL*, pages 214–227. ACM Press, 1987. ISBN 0-89791-215-2.

[10] A. Chapoutot and M. Martel. Abstract simulation: A static analysis of simulink models. In T. Chen, D. N. Serpanos, and W. Taha, editors, *ICESS*, pages 83–92. IEEE, 2009.

[11] A. Gamatié and L. Gonnord. Static analysis of synchronous programs in signal for efficient design of multi-clocked embedded systems. In J. Vitek and B. D. Sutter, editors, *LCTES*, pages 71–80. ACM, 2011.

[12] R. Goldblatt. *Lectures on the Hyperreals: An Introduction to Nonstandard Analysis*. Springer-Verlag, 1998.

[13] I. Hasuo and K. Suenaga. Exercises in *Nonstandard Static Analysis* of hybrid systems. In P. Madhusudan and S. A. Seshia, editors, *CAV*, volume 7358 of *Lect. Notes Comp. Sci.*, pages 462–478. Springer, 2012.

[14] A. E. Hurd and P. A. Loeb. *An Introduction to Nonstandard Real Analysis*. Academic Press, 1985.

[15] G. Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.

[16] N. R. Krishnaswami and N. Benton. Ultrametric semantics of reactive programs. In *LICS*, pages 257–266. IEEE Computer Society, 2011. ISBN 978-0-7695-4412-0.

[17] E. A. Lee and H. Zheng. Operational semantics of hybrid systems. In Morari and Thiele [19], pages 25–53. ISBN 3-540-25108-1.

[18] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995. ISBN 978-0-387-94459-3.

[19] M. Morari and L. Thiele, editors. *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, volume 3414 of *Lecture Notes in Computer Science*, 2005. Springer. ISBN 3-540-25108-1.

[20] H. Nakano. A modality for recursion. In *LICS*, pages 255–266. IEEE Computer Society, 2000. ISBN 0-7695-0725-5.

[21] A. Platzer. *Logical Analysis of Hybrid Systems—Proving Theorems for Complex Dynamics*. Springer, 2010. ISBN 978-3-642-14508-7.

[22] A. Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *Lecture Notes in Computer Science*, pages 446–460. Springer, 2011. ISBN 978-3-642-22437-9.

[23] A. Platzer. The complete proof theory of hybrid systems. In *LICS*, 2012.

[24] A. Robinson. *Non-standard analysis*. Princeton Univ. Press, 1996.

[25] E. Rodríguez-Carbonell and A. Tiwari. Generating polynomial invariants for hybrid systems. In Morari and Thiele [19], pages 590–605. ISBN 3-540-25108-1.

[26] S. Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In K. H. Johansson and W. Yi, editors, *HSCC*, pages 221–230. ACM, 2010. ISBN 978-1-60558-955-8.

[27] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constructing invariants for hybrid systems. *Formal Meth. in Sys. Design*, 32(1): 25–55, 2008.

[28] P. Schrammel and B. Jeannet. From hybrid data-flow languages to hybrid automata: a complete translation. In T. Dang and I. M. Mitchell, editors, *HSCC*, pages 167–176. ACM, 2012. ISBN 978-1-4503-1220-2.

[29] R. Stephens. A survey of stream processing. *Acta Inf.*, 34(7):491–541, 1997.

[30] K. Suenaga and I. Hasuo. Programming with infinitesimals: A while-language for hybrid system modeling. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP (2)*, volume 6756 of *Lect. Notes Comp. Sci.*, pages 392–403. Springer, 2011. ISBN 978-3-642-22011-1.

[31] K. Suenaga, H. Sekine, and I. Hasuo. Hyperstream processing systems: Nonstandard modeling of continuous signal processing. Extended version with proofs, 2013. `www-mmm.is.s.u-tokyo.ac.jp/~ichiro/papers.html`.

[32] T. Terauchi. Dependent types from counterexamples. In M. V. Hermenegildo and J. Palsberg, editors, *POPL*, pages 119–130. ACM, 2010. ISBN 978-1-60558-479-9.

[33] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic. Translating discrete-time simulink to Lustre. *ACM Trans. Embedded Comput. Syst.*, 4(4): 779–818, 2005.

[34] Z. Wan and P. Hudak. Functional reactive programming from first principles. In M. S. Lam, editor, *PLDI*, pages 242–252. ACM, 2000. ISBN 1-58113-199-2.

[35] K. R. Wicks. Nonstandard analysis of ordered sets. *Order*, 12:265–293, 1995.

A. Appendix: Continuity Results for §3.2

Here we collect the continuity results for the denotational semantics of SPROC. These results are mostly straightforward but we present their detailed proofs for the record.

First we review the concrete cpo structure of $\mathbb{C}^\infty$ with the prefix order $\sqsubseteq$ (Lem. 3.4).

Lemma A.1. *Let $s_0 \sqsubseteq s_1 \sqsubseteq \cdots$ be an ascending chain in $\mathbb{C}^\infty$. The stream $\bigsqcup_i s_i \in \mathbb{C}^\infty$, defined by*

$$(\bigsqcup_i s_i)(n) \;:=\; \bigsqcup_i (s_i(n)) \ ,$$

is the supremum of the chain $(s_i)_{i\in\mathbb{N}}$. Here the supremum $\bigsqcup_i$ on the right-hand side is taken in the flat cpo $\mathbb{C}\cup\{\bot\}$. □

The following well-known result is similar to [1, Prop. 2.1.12].

Lemma A.2. *Let X be a cpo; and $x_{i,j} \in X$ be an element, for each $i, j \in \mathbb{N}$, such that*

$$i \le i' \ , \quad j \le j' \quad \Longrightarrow \quad x_{i,j} \sqsubseteq x_{i',j'} \ .$$

Then we have $\bigsqcup_i \bigsqcup_j x_{i,j} = \bigsqcup_j \bigsqcup_i x_{i,j} = \bigsqcup_i x_{i,i}$.

Proof. The direction $\bigsqcup_i \bigsqcup_j x_{i,j} \sqsupseteq \bigsqcup_i x_{i,i}$ is straightforward. For the other direction $\bigsqcup_i \bigsqcup_j x_{i,j} \sqsubseteq \bigsqcup_i x_{i,i}$, one uses the fact that $x_{i,j} \sqsubseteq x_{i,i}$ if $j \le i$ (which follows from the assumption). □

Lemma A.3. *Let $e \in \mathbf{SExp}_{\mathbb{C}}$ and $b \in \mathbf{SExp}_{\mathbb{B}}$ be arbitrary stream expressions. The functions*

$$\begin{aligned}
&[\![e]\!] : \mathbf{SVarEnv} \times \mathbf{NdEnv} \to \mathbb{C}^\infty \ , \quad (J,K) \mapsto [\![e]\!]_{J,K} \ ,\\
&[\![b]\!] : \mathbf{SVarEnv} \times \mathbf{NdEnv} \to \mathbb{B}^\infty \ , \quad (J,K) \mapsto [\![b]\!]_{J,K} \ ,
\end{aligned}$$

are both continuous, with respect to the cpo structures in Lem. 3.4 and 3.6.

Proof. It suffices to prove the continuity in J and K, separately. Let $J_0 \sqsubseteq J_1 \sqsubseteq \cdots$ and $K_0 \sqsubseteq K_1 \sqsubseteq \cdots$ be ascending chains in **SVarEnv** and **NdEnv**, respectively. The proof is by induction on the construction of e and b.

Case $e \equiv x$ with $x \in \mathbf{SVar}$

$$[\![x]\!]_{\bigsqcup J_i,K} = (\bigsqcup_i J_i)(x) = \bigsqcup_i \bigl(J_i(x)\bigr) = \bigsqcup_i [\![x]\!]_{J_i,K} \ .$$

This proves continuity in J. Continuity in K is obvious.

Case $e \equiv c$ with $c \in \mathbb{C}$ Obvious.

Case $e \equiv e_1 \,\mathtt{aop}\, e_2$ with $\mathtt{aop} \in \{+, \times, \hat{}\,\}$ On the continuity in J:

$$\begin{aligned}
&[\![e_1 \,\mathtt{aop}\, e_2]\!]_{\bigsqcup J_i,K}\\
&= \bigl(\, [\![e_1]\!]_{\bigsqcup J_i,K}(n) \ \mathtt{aop}_\bot \ [\![e_2]\!]_{\bigsqcup J_i,K}(n) \,\bigr)_{n\in\mathbb{N}} \quad \text{by Def. 3.7}\\
&= \bigl(\, (\textstyle\bigsqcup_i [\![e_1]\!]_{J_i,K})(n) \ \mathtt{aop}_\bot \ (\bigsqcup_i [\![e_2]\!]_{J_i,K})(n) \,\bigr)_{n\in\mathbb{N}} \quad \text{by I.H.}\\
&= \bigl(\, \textstyle\bigsqcup_i \bigl([\![e_1]\!]_{J_i,K}(n)\bigr) \ \mathtt{aop}_\bot \ \bigsqcup_i \bigl([\![e_2]\!]_{J_i,K}(n)\bigr) \,\bigr)_{n\in\mathbb{N}} \quad \text{by Lem. A.1}\\
&= \bigl(\, \textstyle\bigsqcup_i \bigsqcup_{i'} \bigl([\![e_1]\!]_{J_i,K}(n) \ \mathtt{aop}_\bot \ [\![e_2]\!]_{J_{i'},K}(n)\bigr) \,\bigr)_{n\in\mathbb{N}} \quad (*)\\
&= \bigl(\, \textstyle\bigsqcup_i \bigl([\![e_1]\!]_{J_i,K}(n) \ \mathtt{aop}_\bot \ [\![e_2]\!]_{J_i,K}(n)\bigr) \,\bigr)_{n\in\mathbb{N}} \quad \text{by Lem. A.2, } (\dagger)\\
&= \textstyle\bigsqcup_i \bigl(\, [\![e_1]\!]_{J_i,K}(n) \ \mathtt{aop}_\bot \ [\![e_2]\!]_{J_i,K}(n) \,\bigr)_{n\in\mathbb{N}} \quad \text{by Lem. A.1}\\
&= \textstyle\bigsqcup_i [\![e_1 \,\mathtt{aop}\, e_2]\!]_{J_i,K} \quad \text{by Def. 3.7.}
\end{aligned}$$

Here $(*)$ holds since $\mathtt{aop}_\bot : (\mathbb{C}\cup\{\bot\})^2 \to \mathbb{C}\cup\{\bot\}$ is (obviously) a continuous function with respect to the flat order on $\mathbb{C}\cup\{\bot\}$. In $(\dagger)$, note that continuity implies monotonicity.

The continuity in K is shown in the exactly the same way.

Case $e \equiv e_1 \,\mathtt{fby}^j\, e_2$ We prove the continuity in J; the continuity in K is the same.

If the length of $[\![e_1]\!]_{\bigsqcup J_i,k}$ (which is equal to $\bigsqcup [\![e_1]\!]_{J_i,K}$ by I.H.) is at least j, by Lem. A.1, for some $i_0 \in \mathbb{N}$ we have the length of $[\![e_1]\!]_{J_{i_0},k}$ at least j. In this case we have

$$\begin{aligned}
&[\![e_1 \,\mathtt{fby}^j\, e_2]\!]_{\bigsqcup J_i,K}\\
&= \bigl(\, [\![e_1]\!]_{\bigsqcup J_i,K}(0), \dots, [\![e_1]\!]_{\bigsqcup J_i,K}(j-1),\\
&\qquad [\![e_2]\!]_{\bigsqcup J_i,K}(0), [\![e_2]\!]_{\bigsqcup J_i,K}(1), \dots \bigr) \quad \text{by Def. 3.7}\\
&= \bigl(\, (\textstyle\bigsqcup [\![e_1]\!]_{J_i,K})(0), \dots, (\bigsqcup [\![e_1]\!]_{J_i,K})(j-1),\\
&\qquad (\textstyle\bigsqcup [\![e_2]\!]_{J_i,K})(0), (\bigsqcup [\![e_2]\!]_{J_i,K})(1), \dots \bigr) \quad \text{by I.H.}\\
&= \bigl(\, \textstyle\bigsqcup \bigl([\![e_1]\!]_{J_i,K}(0)\bigr), \dots, \bigsqcup \bigl([\![e_1]\!]_{J_i,K}(j-1)\bigr),\\
&\qquad \textstyle\bigsqcup \bigl([\![e_2]\!]_{J_i,K}(0)\bigr), \bigsqcup \bigl([\![e_2]\!]_{J_i,K}(1)\bigr), \dots \bigr) \quad \text{by Lem. A.1}\\
&= \textstyle\bigsqcup_i \bigl(\, [\![e_1]\!]_{J_i,K}(0), \dots, [\![e_1]\!]_{J_i,K}(j-1),\\
&\qquad [\![e_2]\!]_{J_i,K}(0), [\![e_2]\!]_{J_i,K}(1), \dots \bigr) \quad \text{by Lem. A.1}\\
&= \textstyle\bigsqcup_i [\![e_1 \,\mathtt{fby}^j\, e_2]\!]_{J_i,K} \ . \quad (*)
\end{aligned}$$

Here $(*)$ holds due to Def. 3.7; notice that for $i \ge i_0$, the definition of $[\![e_1 \mathtt{fby}^j e_2]\!]_{J_i,K}$ follows the first case in the definition in Table 2.

If the length k of $[\![e_1]\!]_{\bigsqcup J_i,K}$ is smaller than j, then we have

$$\begin{aligned}
&[\![e_1 \,\mathtt{fby}^j\, e_2]\!]_{\bigsqcup J_i,K}\\
&= \bigl(\, [\![e_1]\!]_{\bigsqcup J_i,K}(0), \dots, [\![e_1]\!]_{\bigsqcup J_i,K}(k-1),\\
&\qquad \bot, \bot, \dots \bigr) \quad \text{by Def. 3.7}\\
&= \bigl(\, \textstyle\bigsqcup \bigl([\![e_1]\!]_{J_i,K}(0)\bigr), \dots, \bigsqcup \bigl([\![e_1]\!]_{J_i,K}(k-1)\bigr),\\
&\qquad \bot, \bot, \dots \bigr) \quad \text{by I.H. and Lem. A.1}\\
&= \textstyle\bigsqcup \bigl(\, [\![e_1]\!]_{J_i,K}(0), \dots, [\![e_1]\!]_{J_i,K}(k-1),\\
&\qquad \bot, \bot, \dots \bigr) \quad \text{by Lem. A.1}\\
&= \textstyle\bigsqcup [\![e_1 \,\mathtt{fby}^j\, e_2]\!]_{J_i,K} \ . \quad (*)
\end{aligned}$$

Here $(*)$ holds since, for sufficiently large i, the length $[\![e_1]\!]_{J_i,K}$ is k.

Case $e \equiv \mathtt{if}\ b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2$ This case is much like when $e \equiv e_1 \,\mathtt{aop}\, e_2$.

Case $e \equiv \mathtt{proj}_k\, f(e_1, \dots, e_m)$ On the continuity in J:

$$\begin{aligned}
&[\![\mathtt{proj}_k\, f(e_1, \dots, e_m)]\!]_{\bigsqcup J_i,K}\\
&= \pi_k \Bigl(\, K(f)\bigl(\, [\![e_1]\!]_{\bigsqcup J_i,K}, \dots, [\![e_m]\!]_{\bigsqcup J_i,K} \,\bigr) \Bigr) \quad \text{by Def. 3.7}\\
&= \pi_k \Bigl(\, K(f)\bigl(\, \textstyle\bigsqcup [\![e_1]\!]_{J_i,K}, \dots, \bigsqcup [\![e_m]\!]_{J_i,K} \,\bigr) \Bigr) \quad \text{by I.H.}\\
&= \pi_k \textstyle\bigsqcup_{i_1} \cdots \bigsqcup_{i_m} K(f)\bigl(\, [\![e_1]\!]_{J_{i_1},K}, \dots, [\![e_m]\!]_{J_{i_m},K} \,\bigr)\\
&\qquad\qquad \text{by continuity of } K(f), \text{ Def. 3.5}\\
&= \pi_k \textstyle\bigsqcup_i K(f)\bigl(\, [\![e_1]\!]_{J_i,K}, \dots, [\![e_m]\!]_{J_i,K} \,\bigr) \quad \text{by Lem. A.2}\\
&= \textstyle\bigsqcup_i \pi_k K(f)\bigl(\, [\![e_1]\!]_{J_i,K}, \dots, [\![e_m]\!]_{J_i,K} \,\bigr) \quad \text{by continuity of } \pi_k\\
&= \textstyle\bigsqcup_i [\![\mathtt{proj}_k\, f(e_1, \dots, e_m)]\!]_{J_i,K} \quad \text{by Def. 3.7.}
\end{aligned}$$

On the continuity in K:

$$\begin{aligned}
&[\![\mathtt{proj}_k\, f(e_1, \dots, e_m)]\!]_{J,\bigsqcup K_i}\\
&= \pi_k \Bigl(\, (\textstyle\bigsqcup K_i)(f)\bigl(\, [\![e_1]\!]_{J,\bigsqcup K_i}, \dots, [\![e_m]\!]_{J,\bigsqcup K_i} \,\bigr) \Bigr) \quad \text{by Def. 3.7}\\
&= \pi_k \textstyle\bigsqcup_i \Bigl(\, K_i(f)\bigl(\, [\![e_1]\!]_{J,\bigsqcup K_i}, \dots, [\![e_m]\!]_{J,\bigsqcup K_i} \,\bigr) \Bigr) \quad \text{by Lem. 3.6}\\
&= \pi_k \textstyle\bigsqcup_i \Bigl(\, K_i(f)\bigl(\, \bigsqcup_{i_1} [\![e_1]\!]_{J,K_{i_1}}, \dots, \bigsqcup_{i_m} [\![e_m]\!]_{J,K_{i_m}} \,\bigr) \Bigr) \quad \text{by I.H.}\\
&= \pi_k \textstyle\bigsqcup_i \bigsqcup_{i_1} \cdots \bigsqcup_{i_m} K_i(f)\bigl(\, [\![e_1]\!]_{J,K_{i_1}}, \dots, [\![e_m]\!]_{J,K_{i_m}} \,\bigr)\\
&\qquad\qquad \text{by continuity of } K_i(f), \text{ Def. 3.5}\\
&= \pi_k \textstyle\bigsqcup_i K_i(f)\bigl(\, [\![e_1]\!]_{J,K_i}, \dots, [\![e_m]\!]_{J,K_i} \,\bigr) \quad \text{by Lem. A.2}\\
&= \textstyle\bigsqcup_i \pi_k K_i(f)\bigl(\, [\![e_1]\!]_{J,K_i}, \dots, [\![e_m]\!]_{J,K_i} \,\bigr) \quad \text{by continuity of } \pi_k\\
&= \textstyle\bigsqcup_i [\![\mathtt{proj}_k\, f(e_1, \dots, e_m)]\!]_{J,K_i} \quad \text{by Def. 3.7.}
\end{aligned}$$

Case $b \equiv \mathtt{true}, \mathtt{false}$ Obvious.

Case $b \equiv b_1 \wedge b_2, \neg b'$ Case $b \equiv e_1 = e_2$ Case $b \equiv \mathtt{isReal}(e)$

Case $b \equiv e_1 < e_2$ These cases are much like when $e \equiv e_1 \,\mathtt{aop}\, e_2$. □

Lemma A.4. *Let $s_1, \dots, s_m \in \mathbb{C}^\infty$, $e'_1, \dots, e'_l \in \mathbf{SExp}_{\mathbb{C}}$, and $K \in \mathbf{NdEnv}$. We define a function $\Phi_{s_1,\dots,s_m;K} : \mathbf{SVarEnv} \to \mathbf{SVarEnv}$ as follows.*

$$\Phi_{s_1,\dots,s_m;K}(J) \;:=\; J \begin{bmatrix} x_1 \mapsto s_1, \ \dots, \ x_m \mapsto s_m, \\ y_1 \mapsto [\![e'_1]\!]_{J,K}, \ \dots, \ y_l \mapsto [\![e'_l]\!]_{J,K} \end{bmatrix} \tag{31}$$

1. $\Phi_{s_1,\dots,s_m;K} : \mathbf{SVarEnv} \to \mathbf{SVarEnv}$ *is continuous.*
2. $\Phi_{s_1,\dots,s_m;K}$ *is continuous in* $s_1, \dots, s_m \in \mathbb{C}^\infty$.
3. $\Phi_{s_1,\dots,s_m;K}$ *is continuous in* K. *Therefore, by Lem. 3.6, the recursive equation (18) has a least solution.*

Proof. 1. Given a chain $J_0 \sqsubseteq J_1 \sqsubseteq \cdots$ in $\mathbf{SVarEnv}$, we have to show

$$\Phi_{\vec{s};K}(\bigsqcup_i J_i) \;=\; \bigsqcup_i \Phi_{\vec{s};K}(J_i) \ ,$$

where the latter $\bigsqcup$ is taken with respect to the order in $\mathbf{SVarEnv}$ (Lem. 3.6). This is proved as follows.

$$\begin{aligned}\big(\Phi_{\vec{s};K}(\bigsqcup_i J_i)\big)(z) &= \begin{cases} s_j & \text{if } z \equiv x_j,\, j \in [1,m] \\ [\![e'_j]\!]_{\bigsqcup_i J_i, K} & \text{if } z \equiv y_j,\, j \in [1,l] \\ (\bigsqcup_i J_i)(z) & \text{otherwise} \end{cases} \\ &= \big(\bigsqcup_i \Phi_{\vec{s};K}(J_i)\big)(z) \ ,\end{aligned}$$

where we also used the fact that the denotation $[\![e'_j]\!]_{J,K}$ is continuous in J (Lem. A.3).

The items 2. and 3. are proved in a similar way. □

Lemma A.5. *Let* $J_{s_1,\dots,s_m;K} \in \mathbf{SVarEnv}$ *be the least solution of the recursive equation*

$$J_{s_1,\dots,s_m;K} \;=\; \Phi_{s_1,\dots,s_m;K}(J_{s_1,\dots,s_m;K}) \ . \tag{32}$$

Here the function $\Phi_{s_1,\dots,s_m;K}$ *is from (31); therefore the above recursive equation (32) is the one (18) parametrized in* $s_1, \dots, s_m$ *and* K.

1. $J_{s_1,\dots,s_m;K}$ *is continuous in* $s_1, \dots, s_m \in \mathbb{C}^\infty$.
2. $J_{s_1,\dots,s_m;K}$ *is continuous in* K.

Proof. 1. For simplicity we assume $m = 1$; the general case is similar. Let $s_0 \sqsubseteq s_1 \sqsubseteq \cdots$ be a chain in $\mathbb{C}^\infty$; we need to show

$$J_{\bigsqcup_i s_i} \;=\; \bigsqcup_i J_{s_i;K} \ .$$

By the construction of a least fixed point in a cpo, $J_{s;K}$ can be explicitly written as

$$J_{s;K} \;=\; \bigsqcup_j (\Phi_{s;K})^j(\bot)$$

for each $s \in \mathbb{C}^\infty$ and $K \in \mathbf{NdEnv}$. This is used in:

$$\begin{aligned}&J_{\bigsqcup_i s_i;K} \\ &= \bigsqcup_j (\Phi_{\bigsqcup_i s_i;K})^j(\bot) \\ &= \bigsqcup_j (\bigsqcup_i \Phi_{s_i;K})^j(\bot) \quad \text{by Lem. A.4.2} \\ &= \bigsqcup_j \big((\bigsqcup_{i_1} \Phi_{s_{i_1};K}) \circ \cdots \circ (\bigsqcup_{i_j} \Phi_{s_{i_j};K}) \big)(\bot) \\ &= \bigsqcup_j \bigsqcup_{i_1} \cdots \bigsqcup_{i_j} \big((\Phi_{s_{i_1};K}) \circ \cdots \circ (\Phi_{s_{i_j};K}) \big)(\bot) \quad \text{by Lem. A.4.1} \\ &= \bigsqcup_j \bigsqcup_i (\Phi_{s_i;K})^j(\bot) \quad \text{by Lem. A.2} \\ &= \bigsqcup_i \bigsqcup_j (\Phi_{s_i;K})^j(\bot) \quad \text{by Lem. A.2} \\ &= \bigsqcup_i J_{s_i;K} \ .\end{aligned}$$

The item 2. is shown similarly, using Lem. A.4.3. □

The following lemma proves Lem. 3.10.

Lemma A.6. *Let* nd *be a node, and* $K \in \mathbf{NdEnv}$. *Then the denotation* $[\![\mathsf{nd}]\!]_K : (\mathbb{C}^\infty)^m \to (\mathbb{C}^\infty)^n$ *is continuous. Moreover,* $[\![\mathsf{nd}]\!]_K$ *is continuous is* K.

Proof. Immediate from Def. 3.8, Lem. A.3 and Lem. A.5. □

Lemma A.7. *Let* $f_1, \dots, f_N, f_{\mathsf{Main}}$ *be node names and* $\mathsf{nd}_1, \dots, \mathsf{nd}_N, \mathsf{nd}_{\mathsf{Main}}$ *be nodes. We define a function* $\Psi : \mathbf{NdEnv} \to \mathbf{NdEnv}$ *as follows.*

$$\Psi(K) \;:=\; K\left[\begin{array}{l} f_1 \mapsto [\![\mathsf{nd}_1]\!]_K,\ \dots,\ f_N \mapsto [\![\mathsf{nd}_N]\!]_K, \\ f_{\mathsf{Main}} \mapsto [\![\mathsf{nd}_{\mathsf{Main}}]\!]_K \end{array}\right]$$

This function Ψ *is continuous. Therefore, by Lem. 3.6, the recursive equation (19) has a least solution.*

Proof. By the pointwise definition of the order of $\mathbf{NdEnv}$, it suffices to show

$$\Psi(\bigsqcup_i K_i)(f) \;=\; \bigsqcup_i \Psi(K_i)(f)$$

for each node name f. When $f \equiv f_j$ for some $j \in [1, K]$,

$$\Psi(\bigsqcup_i K_i)(f_j) = [\![\mathsf{nd}_j]\!]_{\bigsqcup_i K_i} \overset{\text{Lem. A.6}}{=} \bigsqcup_i [\![\mathsf{nd}_j]\!]_{K_i} = \bigsqcup_i \Psi(K_i)(f_j) \ .$$

It is similar when $f = f_{\mathsf{Main}}$. For any other f, we have

$$\Psi(\bigsqcup_i K_i)(f) = (\bigsqcup_i K_i)(f) = \bigsqcup_i K_i(f) = \bigsqcup_i \Psi(K_i)(f) \ .$$

This concludes the proof. □

Lemma A.8. *Let* pg *a program. Its denotation* $[\![\mathsf{pg}]\!] : (\mathbb{C}^\infty)^{m_{\mathsf{Main}}} \to (\mathbb{C}^\infty)^{n_{\mathsf{Main}}}$ *is continuous.*

Proof. The node environment K_0 is the solution of (19) computed in the cpo $\mathbf{NdEnv}$. By Def. 3.5, we have $K_0(f_{\mathsf{Main}})$ continuous. Now

$$\begin{aligned}&K_0(f_{\mathsf{Main}}) \\ &= K_0\left[\begin{array}{l} f_1 \mapsto [\![\mathsf{nd}_1]\!]_{K_0},\ \dots,\ f_N \mapsto [\![\mathsf{nd}_N]\!]_{K_0}, \\ f_{\mathsf{Main}} \mapsto [\![\mathsf{nd}_{\mathsf{Main}}]\!]_{K_0} \end{array}\right](f_{\mathsf{Main}}) \\ &\qquad \text{since } K_0 \text{ is the solution of (19)} \\ &= [\![\mathsf{nd}_{\mathsf{Main}}]\!]_{K_0} \\ &= [\![\mathsf{pg}]\!] \quad \text{by def. of } [\![\mathsf{pg}]\!].\end{aligned}$$

Therefore $[\![\mathsf{pg}]\!]$ is continuous, too. □

B. Appendix: Domain Theory, Transferred

The semantics of SPROC[dt] is most naturally introduced by solving recursive equations on the "domain" of hyperstreams. Here we present necessary theoretical foundations—they are much like in [3, §2.2]—identifying the set of hyperstreams as a *hyperdomain* and *-transferring domain theory.

The definitions and results here are essentially those in [3, §2.2], where what we call a hyperdomain is called an *internal domain*, and a *-continuous function is called an *internal continuous function*. The way we formulate these notions is however a bit different: we favor more explicit use of *-transforms, since this aids deductive verification via the transfer principle.

Definition B.1. In what follows we employ the theory of NSA presented in §2.2. As the base set of a superstructure $V(X)$ (Def. 2.4), we take $X = \mathbb{C} \cup \mathbb{B} \cup \mathbf{SVar} \cup \mathbf{NdName} \cup \mathbf{Var}$.

In Notation 3.3 we adopted the convention to regard a finite stream $(a_0, \dots, a_m)$ as an infinite stream $(a_0, \dots, a_m, \bot, \bot, \dots)$. From now on we take this as the definition of $\mathbb{C}^\infty$:

$$\begin{aligned}\mathbb{C}^\infty := \{\ & s : \mathbb{N} \to \mathbb{C} \cup \{\bot\} \mid \forall n, n' \in \mathbb{N}. \\ & (n \le n' \wedge s(n) = \bot \implies s(n') = \bot)\ \} \ .\end{aligned} \tag{33}$$

We *-transform this set (under the map $^*(\_)$ in (9)) and come to the following notion. A similar notion (without $\bot$) is called *hypersequence* commonly in NSA; see e.g. [12, Chap. 6].

Definition B.2 (Hyperstream)**.** A $\mathbb{C}$-*hyperstream* is an element $s \in {}^*(\mathbb{C}^\infty)$. Similarly, a $\mathbb{B}$-*hyperstream* is an element of ${}^*(\mathbb{B}^\infty)$.

Lemma B.3. *A $\mathbb{C}$-hyperstream is precisely an internal function $s : {}^*\mathbb{N} \to {}^*\mathbb{C} \cup \{\bot\}$ that satisfies*

$$\forall n, n' \in {}^*\mathbb{N}.\, \big(n \le n' \wedge s(n) = \bot \implies s(n') = \bot\big) \ . \tag{34}$$

Proof. By (33) we have the following $\mathscr{L}_\mathbb{C}$-formulas valid.

$$\begin{array}{l} \forall s \in \mathbb{C}^\infty.\, s \in (\mathbb{N} \to \mathbb{C} \cup \{\bot\}) \ , \\ \forall s \in (\mathbb{N} \to \mathbb{C} \cup \{\bot\}).\, \big(s \in \mathbb{C}^\infty \Leftrightarrow \\ \qquad \forall n, n' \in \mathbb{N}.\, \big(n \le n' \wedge s(n) = \bot \Rightarrow s(n') = \bot\big)\big) \end{array}$$

The claim follows immediately by transfer (Lem. 2.9). Recall the notion of internal function (Lem. 2.12), and that ${}^*(\mathbb{C} \cup \{\bot\}) = {}^*\mathbb{C} \cup \{\bot\}$ (Lem. 2.10.4). □

More generally:

Definition B.4 (Hyperdomain)**.** A *hyperdomain* is the pair of *-transforms $({}^*D, {}^*\sqsubseteq)$ of a cpo $(D, \sqsubseteq)$.

Note that ${}^*\sqsubseteq$ is an order in ${}^*D$ (Lem. 2.10.5). This is the notion on which we wish to establish a suitable fixed point property.[3] Towards that goal, we first formulate the definitions of cpo and continuous function as $\mathscr{L}_X$-formulas, so that they can be transferred.

$$\begin{array}{l} \mathsf{BinRel}_{a,r} :\equiv\ r \subseteq a \times a \qquad \mathsf{Refl}_{a,r} :\equiv\ \forall x \in a.\, (x,x) \in r \\ \mathsf{Trans}_{a,r} :\equiv\ \forall x, y, z \in a.\, \big((x,y) \in r \wedge (y,z) \in r \Rightarrow (x,z) \in r\big) \\ \mathsf{AntiSym}_{a,r} :\equiv\ \forall x, y \in a.\, \big((x,y) \in r \wedge (y,x) \in r \Rightarrow x = y\big) \\ \mathsf{Poset}_{a,r} :\equiv\ \mathsf{BinRel}_{a,r} \wedge \mathsf{Refl}_{a,r} \wedge \mathsf{Trans}_{a,r} \wedge \mathsf{AntiSym}_{a,r} \\ \mathsf{HasBot}_{a,r} :\equiv\ \exists x \in a.\, \forall y \in a.\, (x,y) \in r \\ \mathsf{AscCn}_{a,r}(s) :\equiv\ \forall x, x' \in \mathbb{N}.\, (x \le x' \Rightarrow (s(x), s(x')) \in r) \\ \mathsf{UpBd}_{a,r}(b, s) :\equiv\ \forall x \in \mathbb{N}.\, ((s(x), b) \in r) \\ \mathsf{Sup}_{a,r}(p, s) :\equiv\ \mathsf{UpBd}_{a,r}(p, s) \wedge \forall b \in a.\, (\mathsf{UpBd}_{a,r}(b, s) \Rightarrow (p, b) \in r) \end{array}$$

Recall that the inclusion $\mathbb{N} \subseteq X$ is assumed (Def. 2.4). These formulas are used in:

$$\begin{array}{l} \mathsf{CPO}_{a,r} :\equiv\ \mathsf{Poset}_{a,r} \wedge \mathsf{HasBot}_{a,r} \wedge \\ \quad \forall s \in (\mathbb{N} \to a).\, \big(\mathsf{AscCn}_{a,r}(s) \Rightarrow \exists p \in a.\, \mathsf{Sup}_{a,r}(p, s)\big) \ , \\ \mathsf{Conti}_{a_1,r_1,a_2,r_2}(f) :\equiv\ \forall s \in (\mathbb{N} \to a_1).\, \forall p \in a_1. \\ \quad \big(\mathsf{AscCn}_{a_1,r_1}(s) \wedge \mathsf{Sup}_{a_1,r_1}(p, s) \Rightarrow \mathsf{Sup}_{a_2,r_2}(f(p), f \circ s)\big) \ . \end{array} \tag{35}$$

Definition B.5 (*-Continuous function)**.** Let $({}^*D_1, {}^*\sqsubseteq_1)$ and $({}^*D_2, {}^*\sqsubseteq_2)$ be hyperdomains. A function $f : {}^*D_1 \to {}^*D_2$ is **-continuous* if it is internal and satisfies the *-transform of the formula $\mathsf{Conti}_{D_1,\sqsubseteq_1,D_2,\sqsubseteq_2}$. That is to be precise: ${}^*(\mathsf{Conti}_{D_1,\sqsubseteq_1,D_2,\sqsubseteq_2})(f)$ is valid.[4] The set of *-continuous functions is denoted by ${}^*D_1 \to_{{}^*\mathrm{ct}} {}^*D_2$.

Lemma B.6. $({}^*D_1 \to_{{}^*\mathrm{ct}} {}^*D_2) = {}^*(D_1 \to_{\mathrm{ct}} D_2)$. *Here $\to_{\mathrm{ct}}$ denotes the set of continuous functions.*

Proof. Assume $f \in {}^*(D_1 \to_{\mathrm{ct}} D_2)$. The following closed formula is valid in $V(X)$:

$$\forall f' \in (D_1 \to D_2).\, \big(f' \in (D_1 \to_{\mathrm{ct}} D_2) \Leftrightarrow \mathsf{Conti}(f')\big) \ ,$$

where Conti is short for $\mathsf{Conti}_{D_1,\sqsubseteq_1,D_2,\sqsubseteq_2}$. By transfer we have

$$\forall f' \in {}^*(D_1 \to D_2).\, \big(f' \in {}^*(D_1 \to_{\mathrm{ct}} D_2) \Leftrightarrow {}^*\mathsf{Conti}(f')\big) \tag{36}$$

valid in $V({}^*X)$. Thus f satisfies ${}^*\mathsf{Conti}(f')$. Obviously f is internal; therefore $f \in ({}^*D_1 \to_{{}^*\mathrm{ct}} {}^*D_2)$.

Conversely, assume $f \in ({}^*D_1 \to_{{}^*\mathrm{ct}} {}^*D_2)$. By the definition of *-continuity, f is internal, hence by Lem. 2.12 we have $f \in {}^*(D_1 \to D_2)$. Moreover, using the definition of *-continuity and (36), we have $f \in {}^*(D_1 \to_{\mathrm{ct}} D_2)$. □

Lemma B.7. *Let $({}^*D, {}^*\sqsubseteq)$ be a hyperdomain. Then a *-continuous function $f : {}^*D \to {}^*D$ has a least fixed point. Moreover, the function ${}^*\mu$ that maps f to its least fixed point $({}^*\mu)(f)$ is *-continuous.*

Proof. By the usual construction in a cpo, we obtain the map

$$\mu \ : \ (D \to_{\mathrm{ct}} D) \to_{\mathrm{ct}} D \ , \quad f \mapsto \textstyle\bigsqcup_{n \in \mathbb{N}} f^n(\bot) \ .$$

Continuity of μ is easy; its proof is much like that of Lem. A.5. As its *-transform we obtain a function ${}^*\mu : ({}^*D \to_{{}^*\mathrm{ct}} {}^*D) \to_{{}^*\mathrm{ct}} {}^*D$, where we used Lem. B.6 and 2.10. The fact that ${}^*\mu$ returns least fixed points is shown by the transfer of the following $\mathscr{L}_X$-formula.

$$\begin{array}{l} \forall f \in (D \to_{\mathrm{ct}} D).\, \big(f(\mu(f)) = \mu(f) \wedge \\ \qquad\qquad \forall x \in D.\, (f(x) = x \Rightarrow \mu(f) \sqsubseteq x)\big) \end{array}$$ □

Remark B.8. It is crucial in the previous lemma that $f : {}^*D \to {}^*D$ is an internal function. Specifically: recall that a formula A must be closed in order to be transferred (Lem. 2.9); and that in $\mathscr{L}_X$ only bounded quantifiers ($\forall x \in s$ with some bound s) are allowed. Without f being internal, we cannot find such a bound.

C. Appendix: Denotational Semantics of SPROC$^{\mathrm{dt}}$, Alternative Definitions

Here we elaborate on Rem. 4.6 and present a more direct definition of the denotational semantics of SPROC$^{\mathrm{dt}}$. It starts from the same denotations $[\![e]\!]$, $[\![b]\!]$ of hyperstream expressions as in §4.2; but the denotations $[\![\mathsf{nd}]\!]$, $[\![\mathsf{pg}]\!]$ of nodes and programs are obtained more directly by solving "hyperdomain equations," instead of solving domain equations in each section and putting together (as is done in Def. 4.5). We denote the alternative, more direct definitions of denotation by $[\![\mathsf{nd}]\!]'$ and $[\![\mathsf{pg}]\!]'$; and show that they coincide with $[\![\mathsf{nd}]\!]$ and $[\![\mathsf{pg}]\!]$ in §4.2.

Let nd be the following node in SPROC$^{\mathrm{dt}}$.

$$\mathsf{nd} \equiv \left[\begin{array}{l} \texttt{node}\ f(x_1, \dots, x_m)\ \texttt{returns}\ (e_1, \dots, e_n) \\ \texttt{where}\ y_1 = e'_1; \dots ; y_l = e'_l \end{array} \right] \tag{37}$$

In SPROC we have obtained the following map. In particular, its continuity is established in Lem. A.4.

$$\begin{array}{l} \Phi : (\mathbf{SVarEnv} \times \mathbf{NdEnv} \to_{\mathrm{ct}} \mathbb{C}^\infty)^l \to_{\mathrm{ct}} \\ \quad \big((\mathbb{C}^\infty)^m \times \mathbf{NdEnv} \to_{\mathrm{ct}} (\mathbf{SVarEnv} \to_{\mathrm{ct}} \mathbf{SVarEnv})\big) \ , \\ \Phi(f_1, \dots, f_l)(s_1, \dots, s_m, K)(J) := \\ \qquad J \left[\begin{array}{l} x_1 \mapsto s_1, \ \dots, \ x_m \mapsto s_m, \\ y_1 \mapsto f_1(J, K), \ \dots, \ y_l \mapsto f_l(J, K) \end{array} \right] \ . \end{array}$$

Using Lem. B.6 and 2.10, we obtain its *-transform ${}^*\Phi$ as a function of the following type.

$$\begin{array}{l} {}^*\Phi : ({}^*\mathbf{SVarEnv} \times {}^*\mathbf{NdEnv} \to_{{}^*\mathrm{ct}} {}^*(\mathbb{C}^\infty))^l \to_{{}^*\mathrm{ct}} \\ \quad \big({}^*(\mathbb{C}^\infty)^m \times {}^*\mathbf{NdEnv} \to_{{}^*\mathrm{ct}} ({}^*\mathbf{SVarEnv} \to_{{}^*\mathrm{ct}} {}^*\mathbf{SVarEnv})\big) \ . \end{array}$$

By the way, a $\mathbb{C}$-hyperstream expressions $e \in \mathbf{SExp}_\mathbb{C}$ in SPROC$^{\mathrm{dt}}$ determines a function

$$[\![e]\!] : {}^*\mathbf{SVarEnv} \times {}^*\mathbf{NdEnv} \to_{{}^*\mathrm{ct}} {}^*(\mathbb{C}^\infty) \tag{38}$$

by Def. 4.4. We feed ${}^*\Phi$ with thus obtained $[\![e'_1]\!], \dots, [\![e'_l]\!]$:

$$\begin{array}{l} ({}^*\Phi)([\![e'_1]\!], \dots, [\![e'_l]\!]) : \\ \quad {}^*(\mathbb{C}^\infty)^m \times {}^*\mathbf{NdEnv} \to_{{}^*\mathrm{ct}} ({}^*\mathbf{SVarEnv} \to_{{}^*\mathrm{ct}} {}^*\mathbf{SVarEnv}) \ . \end{array}$$

[3] We believe an even more general setting is possible, by defining a hyperdomain to be an internal set $D' \in V({}^*X)$ that satisfies a suitable formula like $\mathsf{CPO}_{a,r}$ in (35). Here we do not need such generality.

[4] We note that the condition is different from (somewhat informal) "${}^*\mathsf{Conti}_{{}^*D_1,{}^*\sqsubseteq_1,{}^*D_2,{}^*\sqsubseteq_2}(f)$ is valid." In the former a chain s ranges over internal functions $s \in {}^*(\mathbb{N} \to D_1)$, while in the latter s can also be external.

Now we postcompose the fixed point operator ${}^*\mu$ from Lem. B.7 and obtain

$$J \;:=\; ({}^*\mu) \circ \big(({}^*\Phi)([\![e'_1]\!], \dots, [\![e'_l]\!])\big) : {}^*(\mathbb{C}^\infty)^m \times {}^*\mathbf{NdEnv} \to_{{}^*\mathrm{ct}} {}^*\mathbf{SVarEnv} \enspace .$$

The value $J(s_1, \dots, s_m, K) \in {}^*\mathbf{SVarEnv}$ thus obtained plays the role of the variable environment J_0 in (18), in the following.

Definition C.1 ($[\![\mathsf{nd}]\!]'$)**.** The denotation

$$[\![\mathsf{nd}]\!]' : {}^*\mathbf{NdEnv} \to_{{}^*\mathrm{ct}} \big({}^*(\mathbb{C}^\infty)^m \to_{{}^*\mathrm{ct}} {}^*(\mathbb{C}^\infty)^n \big)$$

of the node nd in (37), in SPROC$^{\mathsf{dt}}$, is defined by

$$[\![\mathsf{nd}]\!]'(K)(\vec{s}) := \Big([\![e_1]\!]\big(J(\vec{s}, K), K\big), \dots, [\![e_n]\!]\big(J(\vec{s}, K), K\big) \Big) \enspace ,$$

where $[\![e_i]\!]$ is the denotation in Def. 4.4, like (38).

Lemma C.2. *The denotation $[\![\mathsf{nd}]\!]'$ in Def. C.1 coincides with $[\![\mathsf{nd}]\!]$ in Def. 4.5.*

Proof. Let $J_0 \in {}^*\mathbf{SVarEnv}$ be defined by

$$\begin{aligned} J_0 \;&:= J(s_1, \dots, s_m, K) \\ &= ({}^*\mu)\Big(\big(({}^*\Phi)([\![e'_1]\!], \dots, [\![e'_l]\!])\big)(s_1, \dots, s_m, K) \Big) \enspace . \end{aligned}$$

We shall prove that the tuple $\big([\![e_1]\!](J_0, K), \dots, [\![e_l]\!](J_0, K) \big)$—which is precisely $[\![\mathsf{nd}]\!]'(K)(\vec{s})$—coincides with $[\![\mathsf{nd}]\!](K)(\vec{s})$. This amounts to checking the validity of the following closed $\mathscr{L}_{{}^*X}$-formula.

$$\begin{aligned} &\forall s_1, \dots, s_m \in {}^*(\mathbb{C}^\infty).\, \forall K \in {}^*\mathbf{NdEnv}.\, \forall J_0 \in {}^*\mathbf{SVarEnv}. \\ &J_0 = ({}^*\mu)\Big(\big(({}^*\Phi)(M\big[([\![e'_1|_i]\!])_i\big], \dots, M\big[([\![e'_l|_i]\!])_i\big])\big)(\vec{s}, K) \Big) \\ &\Rightarrow \left[\begin{array}{l} \Big(M\big[([\![e_1|_i]\!])_i\big](J_0, K), \dots, M\big[([\![e_n|_i]\!])_i\big](J_0, K)\Big) \\ = M\big[([\![\mathsf{nd}|_i]\!])_i\big](K)(\vec{s}) \end{array} \right] \end{aligned}$$

Here we have used the (sectionwise) definitions of $[\![e_j]\!], [\![e'_j]\!]$ (Def. 4.4) and of $[\![\mathsf{nd}]\!]$ (Def. 4.5). In view of Łoś' theorem (Lem. 2.13), it suffices to show that the following $\mathscr{L}_X$-formula is valid for almost every $i \in \mathbb{N}$.

$$\begin{aligned} &\forall s_1, \dots, s_m \in \mathbb{C}^\infty.\, \forall K \in \mathbf{NdEnv}.\, \forall J_0 \in \mathbf{SVarEnv}. \\ &J_0 = \mu\Big(\Phi([\![e'_1|_i]\!], \dots, [\![e'_l|_i]\!])(\vec{s}, K) \Big) \\ &\Rightarrow \left[\begin{array}{l} \Big([\![e_1|_i]\!](J_0, K), \dots, [\![e_n|_i]\!](J_0, K)\Big) \\ = [\![\mathsf{nd}|_i]\!](K)(\vec{s}) \end{array} \right] \end{aligned}$$

This is nothing but the definition of $[\![\mathsf{nd}|_i]\!]$ (Def. 3.8), the denotation of an SPROC node $\mathsf{nd}|_i$, expressed in $\mathscr{L}_X$. Hence it is valid for each i. □

In exactly the same way an alternative denotation $[\![\mathsf{pg}]\!]'$ of a program in SPROC$^{\mathsf{dt}}$ can be defined, and shown to be equal to the sectionwise semantics $[\![\mathsf{pg}]\!]$ in §4.2.

D. Appendix: Omitted Proofs

D.1 Proof of Lem. 2.10

Proof. 1. For each element b of a we have the $\mathscr{L}_X$-formula $b \in a$ valid. By transfer (Lem. 2.9) we obtain ${}^*b \in {}^*a$ valid in $V({}^*X)$. For injectivity, transfer the $\mathscr{L}_X$-formula $\neg(b = b')$.

2. Let $a = \{b_1, \dots, b_m\}$; then we have

$\forall x \in a.\, (x = b_1 \vee \dots \vee x = b_m)$, valid, thus

$\forall x \in {}^*a.\, (x = {}^*b_1 \vee \dots \vee x = {}^*b_m)$ is valid by transfer (Lem. 2.9).

We leave the rest of the proof (easy by transfer) to [14, §II.3]. □

D.2 Proof of Lem. 2.12

Proof. For the nontrivial 'only if' direction one uses the transfer principle, much like the proof of [14, Lem. 6.7]. □

D.3 Proof of Lem. 3.6

Proof. The proof is easy for $\mathbf{SVarEnv}$. For $\mathbf{NdEnv}$, we have additionally to show that the pointwise supremum of an ascending chain of continuous functions is again continuous. This is shown much like the standard result that the set of continuous functions between cpo's is again a cpo. See e.g. [1, Prop. 2.1.18]. □

D.4 Proof of Thm. 3.25

Here are some usual lemmas. Their proofs are straightforward.

Lemma D.1. *Let $P \in \mathbf{Fml}$, $a \in \mathbf{AExp}$ and $L, L' \in \mathbf{Val}$ (Def. 3.15).*

1. *Assume $L(u) = L'(u)$ for each $u \in \mathrm{FV}(P)$. In this case, $L \models P$ if and only if $L' \models P$.*
2. *$L \models P[a/x]$ if and only if $L[x \mapsto [\![a]\!]_L] \models P$.* □

Proof. (Of Thm. 3.25) The proof is mostly straightforward by induction. Here we show some exemplary cases; the cases (NODE) and (PROG) involve the principle of fixed point induction.

<u>Case (AOP)</u> Let $J \models \Gamma$ and $K \models \Delta$. Using the induction hypothesis for the first assumption of the rule, we have

$$\begin{aligned} &\models \Delta; \Gamma \vdash e_i : \textstyle\prod_{v \in \mathbb{N}} \{u_i \in \mathbb{C} \mid P_i\} \quad \text{for } i = 1, 2, \text{ thus} \\ &[\![e_i]\!]_{J,K} \models \textstyle\prod_{v \in \mathbb{N}} \{u_i \in \mathbb{C} \mid P_i\} \enspace , \quad \text{therefore by Def. 3.17,} \\ &L[v \mapsto n, u_i \mapsto [\![e_i]\!]_{J,K}(n)] \models P_i \text{ for each } n \in \mathbb{N} \text{ and } i = 1, 2. \end{aligned}$$

Since $\mathrm{FV}(P_i) \subseteq \{u_i, v\}$ for $i = 1, 2$, by Lem. D.1.1 we obtain

$$\begin{aligned} &L'_n \models P_i \text{ for each } n \in \mathbb{N} \text{ and } i = 1, 2, \text{ where } L'_n \text{ stands for} \\ &L[v \mapsto n, u_1 \mapsto [\![e_1]\!]_{J,K}(n), u_2 \mapsto [\![e_2]\!]_{J,K}(n), u \mapsto [\![e_1 \;\mathtt{aop}\; e_2]\!]_{J,K}(n)] \enspace . \end{aligned}$$

It holds that $L'_n \models u = u_1 \;\mathtt{aop}\; u_2$ too, due to the definition of $[\![e_1 \;\mathtt{aop}\; e_2]\!]_{J,K}$. Note that in case one of $[\![e_1]\!]_{J,K}(n)$, $[\![e_2]\!]_{J,K}(n)$ and $[\![e_1 \;\mathtt{aop}\; e_2]\!]_{J,K}(n)$ is $\bot$, then the whole valuation L'_n is $\bot$ (Def. 3.15), and it trivially satisfies any type.

By the second assumption of the rule,

$$L'_n \models P_1 \wedge P_2 \wedge (u = u_1 \;\mathtt{aop}\; u_2) \;\Rightarrow P \quad \text{for each } n \in \mathbb{N}.$$

Putting the above facts altogether, we have $L'_n \models P$. Finally, since $\mathrm{FV}(P) \subseteq \{v, u\}$, this is equivalent (by Lem. D.1) to

$$L[v \mapsto n, u \mapsto [\![e_1 \;\mathtt{aop}\; e_2]\!]_{J,K}(n)] \models P \quad \text{for each } n \in \mathbb{N}.$$

This proves the validity of the conclusion of the rule.

<u>Case (FBY^j)</u> Let $J \models \Gamma$ and $K \models \Delta$. Similarly to the case (AOP), we have the following by the induction hypothesis.

$$L[v \mapsto n, u \mapsto [\![e_i]\!]_{J,K}(n)] \models P_i \text{ for } n \in \mathbb{N} \text{ and } i = 1, 2;$$

thus in particular

$$L[v \mapsto n - j, u \mapsto [\![e_2]\!]_{J,K}(n - j)] \models P_2 \text{ for } n \geq j.$$

The latter valuation $L[v \mapsto n - j, u \mapsto [\![e_2]\!]_{J,K}(n - j)]$ is nothing but

$$\big(L[v \mapsto n, u \mapsto [\![e_2]\!]_{J,K}(n - j)]\big) \\ \big[v \mapsto [\![v - j]\!]_{L[v \mapsto n, u \mapsto [\![e_2]\!]_{J,K}(n-j)]}\big];$$

hence by Lem. D.1.2,

$$L[v \mapsto n, u \mapsto [\![e_2]\!]_{J,K}(n - j)] \models P_2[v - j/v] \quad \text{for } n \geq j.$$

We can conclude, from the second assumption of the rule, that

$$\begin{aligned} &L[v \mapsto n, u \mapsto [\![e_1 \;\mathtt{fby}^j\; e_2]\!]_{J,K}(n)] \\ &= \begin{cases} L[v \mapsto n, u \mapsto [\![e_1]\!]_{J,K}(n)] & \text{if } n < j \\ L[v \mapsto n, u \mapsto [\![e_2]\!]_{J,K}(n - j)] & \text{if } n \geq j \end{cases} \end{aligned}$$

satisfies P. (Note again that some denotations being $\bot$ makes P trivially satisfied)

<u>Case (NODE)</u> We use the following result.

Lemma D.2. *Let $s_0 \sqsubseteq s_1 \sqsubseteq \cdots$ be a chain in $\mathbb{C}^\infty$, and $\tau \in$ $\mathbf{SType}_\mathbb{C}$. Then*

$$s_i \models \tau \text{ for each } i \in \mathbb{N} \iff \bigsqcup_i s_i \models \tau \ .$$

Proof. Straightforward, using the cpo structure of $\mathbb{C}^\infty$ (Lem. 3.4) and the definition of $\models$ (Def. 3.17). □

Let $s_i \in \mathbb{C}^\infty$ be a stream such that $s_i \models \tau_i$, for each $i \in [1, m]$. Assume $K \models \Delta$. We need to show

$$\pi_k\big([\![\mathsf{nd}]\!]_K(s_1, \ldots, s_m)\big) \models \tau''_k \quad \text{for each } k \in [1, n], \qquad (39)$$

where nd is the node in the rule. By Def. 3.8, the stream on the left-hand side of (39) is

$$\begin{array}{ll} [\![e_k]\!]_{\bigsqcup_j \Phi^j(\bot), K} & \text{where } \Phi \text{ is the map in (20), that is} \\ \bigsqcup_j [\![e_k]\!]_{\Phi^j(\bot), K} & \text{by the continuity of } [\![\_]\!] \text{ (Lem. A.3).} \end{array}$$

Thus by Lem. D.2 it suffices to show that

$$[\![e_k]\!]_{\Phi^j(\bot), K} \models \tau''_k \quad \text{for each } j \in \mathbb{N}. \qquad (40)$$

In fact we prove that $\Phi^j(\bot) \models \Gamma$ for each $j \in \mathbb{N}$. From this (40) follows, using the induction hypothesis for the third assumption of the rule.

For the base case $j = 0$, we have seen $\bot \models \Gamma$ in Lem. 3.23. For the step case we further distinguish three cases. Firstly, to see $\big(\Phi^{j+1}(\bot)\big)(x_i) \models \Gamma(x_i)$,

$$\begin{array}{l} \big(\Phi^{j+1}(\bot)\big)(x_i) \\ = \Phi\big(\Phi^j(\bot)\big)(x_i) \\ = s_i \quad \text{by def. of } \Phi \text{ (20)} \\ \models \tau_i \equiv \Gamma(x_i) \quad \text{by the assumption on } s_i \text{ and the rule's first assumption.} \end{array}$$

Note that for this proof to work, the first assumption of the (NODE) rule must not be a weaker variant $\Delta; \Gamma \vdash x_i : \tau_i$. Secondly, to see $\big(\Phi^{j+1}(\bot)\big)(y_j) \models \Gamma(y_j)$,

$$\begin{array}{l} \big(\Phi^{j+1}(\bot)\big)(y_j) \\ = [\![e_j]\!]_{\Phi^j(\bot), K} \quad \text{by def. of } \Phi \text{ (20)} \\ \models \Gamma(y_j) \quad \text{by } \Phi^j(\bot) \models \Gamma \text{ (I.H.) and the second assumption.} \end{array}$$

Thirdly, to see $\big(\Phi^{j+1}(\bot)\big)(z) \models \Gamma(z)$ for the other variables z, one immediately sees that $\big(\Phi^{j+1}(\bot)\big)(z) = \bot$, thus it trivially satisfies any type. This concludes the case (NODE).

Case (PROG) Much like the case (NODE). □

D.5 Proof of Lem. 4.11

Proof. The first and second are straightforward by induction on a and P. The last is shown as follows.

$$\begin{array}{l} \models P \\ \iff \forall L \in {}^*\mathbf{Val}.\, L \models P \\ \iff \forall L \in {}^*\mathbf{Val}.\, M\big[(\_ \models P|_i)_{i \in I}\big]\big)(L) = \mathtt{tt} \\ \qquad \text{by the second equivalence} \\ \iff \big(\forall L \in \mathbf{Val}.\, (\_ \models P|_i)(L) = \mathtt{tt}\big) \text{ for almost every } i \\ \qquad \text{by Łoś' theorem (Lem. 2.13)} \\ \iff \models P|_i \text{ for almost every } i \\ \iff M\big[(\models P|_i)_{i \in I}\big] = \mathtt{tt} \quad \text{by Łoś' theorem (Lem. 2.13).} \end{array}$$

□

D.6 SPROC$^{\mathrm{dt}}$: Definition of $\models$ for $\mathbb{B}$-Hyperstream Types and Node Types

Definition D.3. Between a $\mathbb{B}$-hyperstream $t \in {}^*(\mathbb{B}^\infty)$ and a $\mathbb{B}$-hyperstream type $\beta \in \mathbf{SType}_\mathbb{B}$ for SPROC$^{\mathrm{dt}}$, the *satisfaction relation* $t \models \beta$ is defined by: $t \models \prod_{v \in {}^*\mathbb{N}} P$ if

$$\begin{array}{l} \text{for each } n \in {}^*\mathbb{N}\text{:} \quad t(n) = \bot \text{ or} \\ \big(t(n) = \mathtt{tt} \iff L[v \mapsto n] \models P \text{ for each } L \in {}^*\mathbf{Val}\big) \ . \end{array}$$

Between a $*$-continuous function $g : {}^*(\mathbb{C}^\infty)^m \to {}^*(\mathbb{C}^\infty)^n$ and a node type $\nu :\equiv (\tau_1, \ldots, \tau_m) \to (\tau'_1, \ldots, \tau'_n) \in \mathbf{NdType}_{m,n}$, we define $g \models \nu$ to hold if

$$\begin{array}{l} \forall s_1, \ldots, s_m, s'_1, \ldots, s'_n \in {}^*(\mathbb{C}^\infty). \\ \left[\begin{array}{l} s_1 \models \tau_1 \ \wedge \cdots \wedge \ s_m \models \tau_m \ \wedge \ (s'_1, \ldots, s'_n) = g(s_1, \ldots, s_m) \\ \quad \Longrightarrow \quad s'_1 \models \tau'_1 \ \wedge \cdots \wedge \ s'_n \models \tau'_n \end{array}\right] \ . \end{array}$$

D.7 Proof of Lem. 4.19

Proof. The left-hand side means the following $\mathscr{L}_X$-formula is valid.

$$\forall J \in {}^*\mathbf{SVarEnv}. \forall K \in {}^*\mathbf{NdEnv}. \big(J \models \Gamma \wedge K \models \Delta \Rightarrow [\![e]\!]_{J,K} \models \tau\big) \ .$$

By Łoś' theorem (Lem. 2.13) and Def. 4.4, this is equivalent to the right-hand side. □

D.8 Proof of Thm. 4.20

Proof. By induction on the derivation. We only present the (FBY$^{\frac{r}{\mathrm{dt}}}$) case; the other cases are all similar. Let $J \in {}^*\mathbf{SVarEnv}$ and $K \in {}^*\mathbf{NdEnv}$ be such that $J \models \Gamma$ and $K \models \Delta$.

We consider the i-th section of the whole rule instance of (FBY$^{\frac{r}{\mathrm{dt}}}$). It results in the following.

$$\frac{\begin{array}{l} \Delta|_i; \Gamma|_i \vdash e_1|_i : \prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P_1|_i\} \\ \Delta|_i; \Gamma|_i \vdash e_2|_i : \prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P_2|_i\} \\ \models \forall v \in \mathbb{N}. \forall u \in \mathbb{C}. \big((v < r(i+1) \wedge P_1|_i \Rightarrow P|_i) \wedge \\ \qquad (v \geq r(i+1) \wedge P_2|_i[(v - \lceil r(i+1) \rceil)/v] \Rightarrow P|_i) \big) \end{array}}{\Delta|_i; \Gamma|_i \vdash e_1|_i \ \mathtt{fby}^{\lceil r(i+1) \rceil} \ e_2|_i : \prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P|_i\}}$$

Its third assumption is equivalent to the one below, by $r \leq \lceil r \rceil < r + 1$:

$$\frac{\begin{array}{l} \Delta|_i; \Gamma|_i \vdash e_1|_i : \prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P_1|_i\} \\ \Delta|_i; \Gamma|_i \vdash e_2|_i : \prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P_2|_i\} \\ \models \forall v \in \mathbb{N}. \forall u \in \mathbb{C}. \big((v < \lceil r(i+1) \rceil \wedge P_1|_i \Rightarrow P|_i) \wedge \\ \qquad (v \geq \lceil r(i+1) \rceil \wedge P_2|_i[(v - \lceil r(i+1) \rceil)/v] \Rightarrow P|_i) \big) \end{array}}{\Delta|_i; \Gamma|_i \vdash e_1|_i \ \mathtt{fby}^{\lceil r(i+1) \rceil} \ e_2|_i : \prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P|_i\}} \ (*)$$

A crucial observation is that $(*)$ becomes an instance of the rule (FBY^j) for SPROC. Let us now denote by $b_i^{(1)} \in \mathbb{B}$ the validity of the first assumption of $(*)$, that is, $b_i^{(1)} = \mathtt{tt}$ iff $\Delta|_i; \Gamma|_i \vdash e_1|_i : \prod_{v \in \mathbb{N}} \{u \in \mathbb{C} \mid P_1|_i\}$ is valid. Similarly, we let $b_i^{(2)}, b_i^{(3)}$ and $b_i^{(4)}$ denote the validity of the second and third assumptions and the conclusion of $(*)$, respectively. Since $(*)$ is a rule instance of SPROC, by its soundness (Thm. 3.25) we have $b_i^{(1)} \wedge b_i^{(2)} \wedge b_i^{(3)} \Rightarrow b_i^{(4)}$ hold. This holds for each $i \in I$; thus by Lem. 2.13,

$$M\big[(b_i^{(1)})_{i \in I}\big] \wedge M\big[(b_i^{(2)})_{i \in I}\big] \wedge M\big[(b_i^{(3)})_{i \in I}\big] \Rightarrow M\big[(b_i^{(4)})_{i \in I}\big] \qquad (41)$$

is valid.

The induction hypothesis says $\models \Delta; \Gamma \vdash e_1 : \prod_{v \in {}^*\mathbb{N}} \{u \in {}^*\mathbb{C} \mid P_1\}$; by Lem. 4.19 this means $M\big[(b_i^{(1)})_{i \in I}\big] = \mathtt{tt}$. By the same argument we have $M\big[(b_i^{(2)})_{i \in I}\big] = \mathtt{tt}$; moreover $M\big[(b_i^{(3)})_{i \in I}\big] = \mathtt{tt}$ by the induction hypothesis and the transfer principle (Lem. 2.9). Therefore by (41) we have $M\big[(b_i^{(4)})_{i \in I}\big] = \mathtt{tt}$, which by Lem. 4.19 implies the validity of $\Delta; \Gamma \vdash e_1 \ \mathtt{fby}^{\frac{r}{\mathrm{dt}}} \ e_2 : \prod_{v \in {}^*\mathbb{N}} \{u \in {}^*\mathbb{C} \mid P\}$, the conclusion of the (FBY$^{\frac{r}{\mathrm{dt}}}$) rule. □

D.9 Proof of Lem. 5.2

Proof. Closure under right derivation is immediate from Def. 5.1.

For the Riemann integration we use the Lebesgue characterization of Riemann integrability:

> f is Riemann integrable in $[0, t]$ if and only if
> $$D_f := \{d \in [0, t] \mid f \text{ is discontinuous at } d\}$$
> has the Lebesgue measure 0.

We shall prove that, for a signal $f \in \mathbf{Signals}$, D_f is a countable set and hence is of measure 0. Indeed, by Def. 5.1, for each $d \in D_f$

there is an interval $(d, d+\varepsilon_d)$ in which f is continuous. Then we have $D_f \cap (d, d+\varepsilon_d) = \emptyset$; and that $(d, d+\varepsilon_d) \cap (d', d'+\varepsilon_{d'}) = \emptyset$ for $d, d' \in D_f$ such that $d \neq d'$. Now an open interval $(d, d+\varepsilon_d)$ necessarily contains a rational number q_d; in this way we obtain an injection $D_f \hookrightarrow \mathbb{Q}$, $d \mapsto q_d$. Therefore $|D_f| \leq |\mathbb{Q}| = \aleph_0$.

We are yet to show that the function $g(t) := \int_0^t f(t')\mathrm{d}t'$ is a signal. We first observe that g is right differentiable and its right derivative g^r coincides with f; this is shown in the same way as the fundamental theorem of calculus is proved.

To show that g has left limits at each $t \in \mathbb{R}_{>0}$, since f has a left limit $L := \lim_{t' \to t-0} f(t')$ there is some $\delta > 0$ such that

$$\forall t'' \in (t-\delta, t).\, f(t'') \in (L-1, L+1) \enspace .$$

Then, for each $t'' \in (t-\delta, t)$,

$$\begin{aligned} & g(t) - g(t'') = \int_0^t f(t')\mathrm{d}t' - \int_0^{t''} f(t')\mathrm{d}t' \\ & = \int_{t''}^t f(t')\mathrm{d}t' \ \in \big((t-t'')(L-1),\, (t-t'')(L+1)\big) \enspace . \end{aligned}$$

As t'' tends to t, the bounds $(t-t'')(L-1)$ and $(t-t'')(L+1)$ both tend to 0. Therefore g is in fact left continuous; in particular it has left limits.

We are done if we show the last condition in Def. 5.1, that is, that g is of class C^∞ in some open interval on the right of each $t \in \mathbb{R}_{\geq 0}$. Take $\varepsilon > 0$ such that f is of class C^∞ in $(t, t+\varepsilon)$. We claim that g is of class C^∞ in the interval $(t, t+\frac{\varepsilon}{2})$. Indeed, since f is right continuous, f is continuous in the closed interval $[t, t+\frac{\varepsilon}{2}]$. This allows us to appeal to the fundamental theorem of calculus: it yields that the mapping

$$h(t'') := \int_t^{t''} f(t')\mathrm{d}t'$$

is differentiable in $[t, t+\frac{\varepsilon}{2}]$ and that its derivative $h'(t'')$ is $f(t'')$. Since f is of class C^∞ in $(t, t+\frac{\varepsilon}{2})$; therefore h is of class C^∞ in $(t, t+\frac{\varepsilon}{2})$. The claim follows since $g(t'') = h(t'') + \int_0^t f(t')\mathrm{d}t'$. □

D.10 Proof of Thm. 5.6

Proof. It is well-known (see [12, 14]) that, if a sequence $a_0, a_1, \dots$ has a limit, then $\mathrm{sh}(M[(a_i)_{i\in I}]) = \lim_{i\to\infty} a_i$. This is used in:

$$\begin{aligned} & \mathrm{sh}\big(\mathsf{Smth}(\mathsf{Smp}(f))(t)\big) \\ & = \mathrm{sh}\Big(\big(M\big[\,\big((f(\tfrac{j}{i+1}))_{j\in\mathbb{N}}\big)_{i\in I}\,\big]\big)\big(M\big[\,(\lceil (i+1)t\rceil)_{i\in I}\,\big]\,\big)\Big) \\ & \qquad \text{by def. of } \mathsf{Smth} \text{ and } \mathsf{Smp} \\ & = \mathrm{sh}\Big(\big(M\big[\,\big(f(\tfrac{\lceil (i+1)t\rceil}{i+1})\big)_{i\in I}\,\big]\big)\Big) \quad \text{by Cor. 2.14} \\ & = \lim_{i\to\infty} f(\tfrac{\lceil (i+1)t\rceil}{i+1}) \quad (*) \\ & = f(t) \enspace , \end{aligned}$$

where $(*)$ holds because $f \in \mathbf{Signals}$ is right continuous and $\frac{\lceil (i+1)t\rceil}{i+1} \to t$ as $i \to \infty$. □

D.11 Proof of Thm. 5.12

Lemma D.4. *1. Let $P \in \mathbf{Fml}$ and $\mathrm{FV}(P) \subseteq \{u, w\}$. The subset*

$$S_P \ := \ \{(a,b) \mid \ \forall L \in \mathbf{Val}.\ L[u\mapsto a, w\mapsto b] \models P\} \ \subseteq \mathbb{C}^2$$

is closed with respect to the usual Euclidean topology. In particular: assuming $\lim_j a_j = a'$, $\lim_j b_j = b'$, and $L[u \mapsto a_j, v \mapsto b_j] \models P$ for each $j \in \mathbb{N}$, we have $L[u \mapsto a', v \mapsto b'] \models P$.

2. Assume that $L[u \mapsto a_i, v \mapsto b_i] \models P$ for almost every $i \in I = \mathbb{N}$, $a = \mathrm{sh}(M[(a_i)_i])$ and $b = \mathrm{sh}(M[(b_i)_i])$. Then we have $L[u \mapsto a, v \mapsto b] \models P$.

Proof. 1. By induction on $P \in \mathbf{Fml}$. We do the case $P \equiv \forall v \in \mathbb{C}.\ P'$; the other cases are easy. We observe that

$$S_P \ = \ \mathbb{C}^m \setminus \pi[\mathbb{C}^{m+1} \setminus S_{P'}] \enspace ,$$

where $\pi : \mathbb{C}^{m+1} \to \mathbb{C}^m$ is a projection (that corresponds to the quantifier $\exists v \in \mathbb{C}$). This projection π is known to be an open map (but not necessarily a closed map). Since $S_{P'}$ is closed (by the induction hypothesis), S_P is closed too.

2. We use 1.; for that we construct suitable converging subsequences $(a_{i_j})_j$ and $(b_{i_j})_j$ in the following inductive way. Let us denote, for each $j \in \mathbb{N}$,

$$\begin{aligned} & S_j := \big\{ i \in \mathbb{N} \mid |a_i - a| < \tfrac{1}{j+1} \wedge |b_i - b| < \tfrac{1}{j+1} \big\} \enspace ; \\ & T := \big\{ i \in \mathbb{N} \mid \forall L \in \mathbf{Val}.\, L[u\mapsto a_i, v\mapsto b_i] \models P \big\} \enspace ; \\ & U_{j+1} := \{ i \in \mathbb{N} \mid i_j < i \} \enspace . \end{aligned}$$

By the definition of shadow (Def. 2.2) $S_j \in \mathcal{F}$ for each j; and $T \in \mathcal{F}$ by the assumption. Each U_j is a cofinite set (§2.1) hence belongs to $\mathcal{F}$, too. We define

$$i_0 := \min(S_0 \cap T) \ ; \quad i_{j+1} := \min(S_{j+1} \cap T \cap U_{j+1}) \enspace .$$

Note that the sets $S_0 \cap T$ and $S_{j+1} \cap T \cap U_{j+1}$ belong to $\mathcal{F}$, thus they are nonempty. The sequence $(i_j)_j = (i_0, i_1, \dots)$ thus obtained satisfies

- $\forall L \in \mathbf{Val}.\ L[u \mapsto a_{i_j}, v \mapsto b_{i_j}] \models P$, since $i_j \in T$;
- $\lim_j a_{i_j} = a$, since $|a_{i_j} - a| < \frac{1}{j+1}$ holds due to $i_j \in S_j$; and
- $\lim_j b_{i_j} = b$ for the same reason.

Therefore by the item 1., we have $L[u \mapsto a, v \mapsto b] \models P$. □

Proof. (Of Thm. 5.12) By soundness of the SPROC$^{\mathtt{dt}}$ type system (Thm. 4.20) we have $\models\vdash \mathsf{pg} : () \to (\tau^{\mathrm{HS}})$, that is, $[\![\mathsf{pg}]\!] \models \tau^{\mathrm{HS}}$ where $\tau^{\mathrm{HS}} \equiv \prod_{v\in{}^*\mathbb{N}} \{u \in \mathbb{C} \mid P^{\mathrm{HS}}[(v \times \mathtt{dt})/w]\}$. This means, by Def. 4.12:

$$\forall n \in {}^*\mathbb{N}.\, \forall L \in {}^*\mathbf{Val}.\, L\big[v \mapsto n, u \mapsto [\![\mathsf{pg}]\!](n)\big] \models P^{\mathrm{HS}}[(v \times \mathtt{dt})/w]$$

By a "sectionwise lemma" (Lem. 4.11) this is equivalent to the following, where we also used $P^{\mathrm{HS}}|_i \equiv P$ (obvious).

$$\begin{aligned} & \forall n \in {}^*\mathbb{N}.\, \forall L \in {}^*\mathbf{Val}. \\ & \Big(M\big[(\_ \models P[(\tfrac{v}{i+1})/w])_i\big]\Big)\Big(L\big[v \mapsto n, u \mapsto [\![\mathsf{pg}]\!](n)\big]\Big) = \mathtt{tt} \end{aligned}$$

Now we use Łoś' theorem (Lem. 2.13)—together with Def. 4.9 of valuation update and Def. 4.5 of $[\![\mathsf{pg}]\!]$—to see that the condition is equivalent to the following one: for almost every $i \in I$,

$$\forall n \in \mathbb{N}.\, \forall L \in \mathbf{Val}.\, L\big[v \mapsto n, u \mapsto [\![\mathsf{pg}|_i]\!](n)\big] \models P[(\tfrac{v}{i+1})/w] \enspace .$$

By Lem. D.1, this is equivalent to the following: for almost every $i \in I$,

$$\forall n \in \mathbb{N}.\, \forall L \in \mathbf{Val}.\, L\big[w \mapsto \tfrac{n}{i+1}, u \mapsto [\![\mathsf{pg}|_i]\!](n)\big] \models P \enspace .$$

In particular, by setting $n = \lceil (i+1)t \rceil$ for each i, we obtain: for almost every $i \in I$,

$$\forall L \in \mathbf{Val}.\, L\big[w \mapsto \tfrac{\lceil (i+1)t\rceil}{i+1}, u \mapsto [\![\mathsf{pg}|_i]\!](\lceil (i+1)t \rceil)\big] \models P \enspace .$$

At this point we use Lem. D.4.2 and obtain:

$$\begin{aligned} & \forall L \in \mathbf{Val}.\ L\Big[w \mapsto \mathrm{sh}\big(M\big[\big(\tfrac{\lceil (i+1)t\rceil}{i+1}\big)_i\big]\big), \\ & \qquad u \mapsto \mathrm{sh}\big(M\big[\big([\![\mathsf{pg}|_i]\!](\lceil (i+1)t\rceil)\big)_i\big]\big)\Big] \models P \enspace . \end{aligned} \tag{42}$$

Now, since $\frac{\lceil (i+1)t\rceil}{i+1}$ converges to t we have $\mathrm{sh}\big(M\big[\big(\frac{\lceil (i+1)t\rceil}{i+1}\big)_i\big]\big) = t$ (see §2.1). Moreover,

$$\begin{aligned} & \mathrm{sh}\big(M\big[\big([\![\mathsf{pg}|_i]\!](\lceil (i+1)t\rceil)\big)_i\big]\big) \\ & = \mathrm{sh}\Big(\Big(M\big[\big([\![\mathsf{pg}|_i]\!]\big)_i\big]\Big)\Big(M\big[\big(\lceil (i+1)t\rceil\big)_i\big]\Big)\Big) \quad \text{by Cor. 2.14} \\ & = \mathrm{sh}\Big([\![\mathsf{pg}]\!]\Big(M\big[\big(\lceil (i+1)t\rceil\big)_i\big]\Big)\Big) \quad \text{by Def. 4.5} \\ & = \mathrm{sh}\Big(\mathsf{Smth}([\![\mathsf{pg}]\!])(t)\Big) \quad \text{by Def. 5.5} \\ & = f(t) \quad \text{by the assumption that } \mathsf{pg} \text{ is a model of } f. \end{aligned}$$

Therefore (42) is equivalent to

$$\forall L \in \mathbf{Val}.\ L[w \mapsto t, u \mapsto f(t)] \models P\ .$$

This proves the claim. □

D.12 Derivation of Assumption (a) of (29)

It is derived as shown in Table 6. The "arithmetic facts" there hold also for hypernumbers, by the transfer (Lem. 2.9) of the corresponding results for standard numbers.

(d)	$\Delta_0; \Gamma_{\mathsf{inv}} \vdash s : \tau_{s\text{-}\mathsf{inv}}$ by (SVAR); note $\Gamma_{\mathsf{inv}}(s) \equiv \tau_{s\text{-}\mathsf{inv}}$
(e)	$\Delta_0; \Gamma_{\mathsf{inv}} \vdash c : \tau_{c\text{-}\mathsf{inv}}$ by (SVAR)
(f)	$\Delta_0; \Gamma_{\mathsf{inv}} \vdash \mathtt{dt} : \prod_{v \in {}^*\mathbb{N}} \{u \in {}^*\mathbb{C} \mid u = \mathtt{dt}\}$ by $(\mathtt{dt})$, Def. 4.16
(g)	$\models \forall v \in {}^*\mathbb{N}. \forall u, u_1, u_2 \in {}^*\mathbb{C}. \big(u_1 = \frac{1}{2}(1 - \mathrm{i} \cdot \mathtt{dt})^v + \frac{1}{2}(1 + \mathrm{i} \cdot \mathtt{dt})^v \wedge u_2 = \mathtt{dt} \wedge u = u_1 \times u_2 \Rightarrow u = \frac{1}{2}\mathtt{dt}(1 - \mathrm{i} \cdot \mathtt{dt})^v + \frac{1}{2}\mathtt{dt}(1 + \mathrm{i} \cdot \mathtt{dt})^v\big)$ an arithmetic fact
(h)	$\Delta_0; \Gamma_{\mathsf{inv}} \vdash c \times \mathtt{dt} : \prod_{v \in {}^*\mathbb{N}} \{u \in {}^*\mathbb{C} \mid u = \frac{1}{2}\mathtt{dt}(1 - \mathrm{i} \cdot \mathtt{dt})^v + \frac{1}{2}\mathtt{dt}(1 + \mathrm{i} \cdot \mathtt{dt})^v\}$ by (e–g) and (AOP)
(i)	$\models \forall v \in {}^*\mathbb{N}. \forall u, u_1, u_2 \in {}^*\mathbb{C}. \big(u_1 = \frac{1}{2}\mathrm{i}(1 - \mathrm{i} \cdot \mathtt{dt})^v - \frac{1}{2}\mathrm{i}(1 + \mathrm{i} \cdot \mathtt{dt})^v \wedge u_2 = \frac{1}{2}\mathtt{dt}(1 - \mathrm{i} \cdot \mathtt{dt})^v + \frac{1}{2}\mathtt{dt}(1 + \mathrm{i} \cdot \mathtt{dt})^v \wedge u = u_1 + u_2 \Rightarrow u = \frac{1}{2}\mathrm{i}(1 - \mathrm{i} \cdot \mathtt{dt})^{v+1} - \frac{1}{2}\mathrm{i}(1 + \mathrm{i} \cdot \mathtt{dt})^{v+1}\big)$ an arithmetic fact
(j)	$\Delta_0; \Gamma_{\mathsf{inv}} \vdash s + c \times \mathtt{dt} : \prod_{v \in {}^*\mathbb{N}} \{u \in {}^*\mathbb{C} \mid u = \frac{1}{2}\mathrm{i}(1 - \mathrm{i} \cdot \mathtt{dt})^{v+1} - \frac{1}{2}\mathrm{i}(1 + \mathrm{i} \cdot \mathtt{dt})^{v+1}\}$ by (d),(h–i) and (AOP)
(k)	$\Delta_0; \Gamma_{\mathsf{inv}} \vdash 0 : \prod_{v \in {}^*\mathbb{N}} \{u \in {}^*\mathbb{C} \mid u = 0\}$ by (CONST)
(l)	$\models \forall v \in {}^*\mathbb{N}. \forall u \in {}^*\mathbb{C}.$ $\big(v < 1 \wedge u = 0 \Rightarrow u = \frac{1}{2}\mathrm{i}(1 - \mathrm{i} \cdot \mathtt{dt})^v - \frac{1}{2}\mathrm{i}(1 + \mathrm{i} \cdot \mathtt{dt})^v\big) \wedge$ $\Big(v \leq 1 \wedge u = \big(\frac{1}{2}\mathrm{i}(1 - \mathrm{i} \cdot \mathtt{dt})^{v+1} - \frac{1}{2}\mathrm{i}(1 + \mathrm{i} \cdot \mathtt{dt})^{v+1}\big)[v - 1/v]$ $\Rightarrow u = \frac{1}{2}\mathrm{i}(1 - \mathrm{i} \cdot \mathtt{dt})^v - \frac{1}{2}\mathrm{i}(1 + \mathrm{i} \cdot \mathtt{dt})^v\Big)$ an arithmetic fact
(m)	$\Delta_0; \Gamma_{\mathsf{inv}} \vdash 0\ \mathtt{fby}^1\ (s + c \times \mathtt{dt}) : \tau_{s\text{-}\mathsf{inv}}$ by (k–l) and (FBY^1)

Table 6. Derivation of the assumption (a)

D.13 Proof of (30)

Proof. By Lem. 2.13 it suffices to show that the following holds for almost every $i \in I = \mathbb{N}$.

$$\models \forall v \in \mathbb{N}. \forall u \in \mathbb{C}. \Big(u = \frac{1}{2}\mathrm{i}(1 - \frac{\mathrm{i}}{i+1})^v - \frac{1}{2}\mathrm{i}(1 + \frac{\mathrm{i}}{i+1})^v \Rightarrow (t_0(i+1) \leq v \vee u \leq 1 + \varepsilon)\Big)\ , \quad \text{that is}$$

$$\models \forall v \in \mathbb{N}. \Big(v < t_0(i+1) \Rightarrow \frac{1}{2}\mathrm{i}(1 - \frac{\mathrm{i}}{i+1})^v - \frac{1}{2}\mathrm{i}(1 + \frac{\mathrm{i}}{i+1})^v \leq 1 + \varepsilon\Big)\ . \tag{43}$$

The number $\frac{1}{2}\mathrm{i}(1 - \frac{\mathrm{i}}{i+1})^v - \frac{1}{2}\mathrm{i}(1 + \frac{\mathrm{i}}{i+1})^v$ is the real part of $\mathrm{i}(1 - \frac{\mathrm{i}}{i+1})^v$, hence is real. Its value is bounded as follows.

$$\begin{aligned}
&\frac{1}{2}\mathrm{i}(1 - \frac{\mathrm{i}}{i+1})^v - \frac{1}{2}\mathrm{i}(1 + \frac{\mathrm{i}}{i+1})^v \\
&= \mathrm{Re}(\mathrm{i}(1 - \frac{\mathrm{i}}{i+1})^v) \leq \|\mathrm{i}(1 - \frac{\mathrm{i}}{i+1})^v\| = \|(1 - \frac{\mathrm{i}}{i+1})\|^v \\
&\leq \left(1 + \frac{1}{(i+1)^2}\right)^{\frac{v}{2}} \leq \left(1 + \frac{1}{(i+1)^2}\right)^{\frac{t_0(i+1)}{2}} \\
&= \left(\left(1 + \frac{1}{(i+1)^2}\right)^{i+1}\right)^{\frac{t_0}{2}} \\
&= \left(\sum_{k=0}^{i+1} \binom{i+1}{k} \frac{1}{(i+1)^{2k}}\right)^{\frac{t_0}{2}} \\
&= \Big(1 + (i+1)\frac{1}{(i+1)^2} + \frac{i(i-1)}{2}\frac{1}{(i+1)^4} \\
&\qquad + \frac{i(i-1)(i-2)}{3!}\frac{1}{(i+1)^6} + \cdots\Big)^{\frac{t_0}{2}} \\
&\leq \left(1 + \frac{1}{(i+1)} + \frac{1}{(i+1)^2} + \frac{1}{(i+1)^3} + \cdots + \frac{1}{(i+1)^{i+1}}\right)^{\frac{t_0}{2}} \\
&\leq \left(1 + \frac{1}{(i+1)} + \frac{i}{(i+1)^2}\right)^{\frac{t_0}{2}} \\
&\xrightarrow{i \to \infty} 1\ .
\end{aligned}$$

Therefore for a large enough i we have (43) hold; hence it holds for almost every i (cf. §2.1). □