Journal of Universal Computer Science, vol. 16, no. 18 (2010), 2629-2656
submitted: 14/12/09, accepted: 18/7/10, appeared: 28/9/10 © J.UCS

Compositional Semantics of Dataflow Networks with Query-Driven Communication of Exact Values[1]

Michal Konečný
(School of Engineering and Applied Science, Aston University, United Kingdom
m.konecny@aston.ac.uk)

Amin Farjudian
(School of Engineering and Applied Science, Aston University, United Kingdom
a.farjudian@aston.ac.uk)

Abstract: We develop and study the concept of *dataflow process networks* as used for example by Kahn to suit exact computation over data types related to real numbers, such as continuous functions and geometrical solids. Furthermore, we consider communicating these exact objects among processes using protocols of a query-answer nature as introduced in our earlier work. This enables processes to provide valid approximations with certain accuracy and focusing on certain locality as demanded by the receiving processes through queries.

We define domain-theoretical denotational semantics of our networks in two ways: (1) *directly*, i. e. by viewing the whole network as a composite process and applying the process semantics introduced in our earlier work; and (2) *compositionally*, i. e. by a fixed-point construction similar to that used by Kahn from the denotational semantics of individual processes in the network. The direct semantics closely corresponds to the operational semantics of the network (i. e. it is correct) but very difficult to study for concrete networks. The compositional semantics enables compositional analysis of concrete networks, assuming it is correct.

We prove that the compositional semantics is a *safe* approximation of the direct semantics. We also provide a method that can be used in many cases to establish that the two semantics *fully coincide*, i. e. safety is not achieved through inactivity or meaningless answers.

The results are extended to cover recursively-defined infinite networks as well as nested finite networks.

A robust prototype implementation of our model is available.

Key Words: exact real computation, distributed computation, dataflow networks, denotational semantics, domain theory
Category: F.1.1, C.2.4, F.3.2, G.1.0, G.0

1 Introduction

In our previous work [Konečný and Farjudian 2010] we introduced the notions of query-answer protocol for communicating partial information about exact values and we gave such protocols a lattice semantics that captures progress of information transfer measured using the lattice of queries. We also formalised the notion of a process that is capable of communicating with other processes using query-answer protocols over its input and output sockets. The behaviour of such a process is captured as a set of possible

[1] Supported by the EPSRC grant number EP/C01037X/1

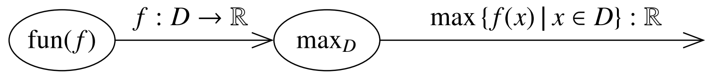

Figure 1: A simple network computing the maximum of a function f over an interval J.

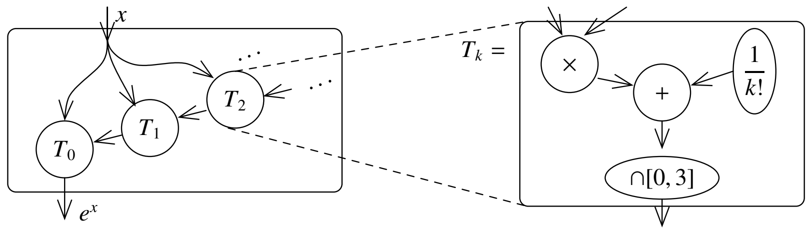

Figure 2: An infinite network computing the exact exponential on [0, 1], cf. [Potts 1998, p.124].

event sets (called traces). From such sets of traces we derived for a process a functional output-to-input query semantics (Q-semantics) and an input-to-output query-answer semantics (QA-semantics).

In this article we build on these notions and study networks of processes, adopting and extending the notation introduced in [Konečný and Farjudian 2010]. Intuitively, sockets are the gates of processes to the outside world, while channels are the roads linking these gates. A collection of processes connected by channels forms a *network*. After hiding all channels a network can be seen as a process, which we call a *composite process*. We formalise this idea first for finite networks then for infinite recursively defined networks, providing means to reason about the semantics and operation of the resulting composite processes.

1.1 Example networks

In [Konečný and Farjudian 2010] we have motivated the study of processes using an informal presentation of two process networks — a small finite network in which there is a process that computes the maximum of a specific continuous function communicated over a channel (Fig 1) and an infinite network that embodies a Taylor expansion of the exponential function (Fig 2). Here we introduce further three examples:

– A network embodying the continuous fraction expansion of the square root (Fig. 4), which is simple yet non-trivial and has a loop. Some processes used in this network are specified using activity diagrams in Fig. 3. The remaining processes (or very similar ones) have been formalised in [Konečný and Farjudian 2010].

- A network embodying the Picard operator (Fig. 5) solving a well-conditioned initial value problem (IVP), which has a loop made of continuous function channels.
- A network parallelising an arbitrary interval-based IVP/BVP solver (Fig. 6), which, unlike the other examples, makes substantial use of parallelism.

Please note that the example networks do not give very efficient ways of solving the associated problems. We use these networks for illustrating particular aspects of our approach and we believe the patterns in them are present also in many examples that have practical value. Such examples are work in progress.

We now give an informal explanation of the last one of the example networks above. Assume that we are given a process called "solver$_D$" that takes as its first input an enclosure of the solution over an interval $D \subseteq \mathbb{R}$. This enclosure is usually very loose except in the origin where it specifies an initial condition. Another input of the solver is an encoding of the ordinary differential equation (in the form of a vector field), possibly combined with boundary condition(s). This information is expressed as a transformer on enclosures of both the solution and its derivative. When given some enclosures, it returns an improved pair of enclosures, typically inferring information about the derivative from the given information about the solution using the differential equation. Boundary conditions are typically used to translate information about the solution from one region of the time domain to another.

To parallelise this given solver, we first need to wrap it as a process called "E-solver$_D$" (E for endpoints) that can communicate not only the solution as a whole but also the values of the solution function at the endpoints of the time domain D. The composite process called "S-solver$_D$" (S for split) shows how the domain D is split into D_1 and D_2 and solved in parallel by two instances of "P-solver$_{D_i}$" (P for parallel), which is a union of one "E-solver$_{D_i}$" and one "S-solver$_{D_i}$" for further splitting.

This recursive definition defines a "P-solver$_D$" as an infinite network with an instance of "E-solver$_{D'}$" for each interval D' within D created by some binary splitting. Each "E-solver$_{D'}$" can be made redundant by instead using the "S-solver$_{D'}$" within the parent "P-solver$_{D'}$", which results in solving the problem in parallel on two halves of the domain of that "E-solver$_{D'}$".

Similar to the Taylor series network, one can derive the semantics of the infinite network as a limit of the semantics of its finite portions (as formally defined in Section 7) and when executing, only a finite portion of the network will be built and activated at each moment.

1.2 Overview of formalisation

In Section 2 we formally define finite process networks and their event traces and in Section 3 we extend Q-semantics and QA-semantics from processes to such networks using a fixed point construction similar to Kahn's. Section 4 defines a composite process and a restriction of the network semantics to the composite process. Most importantly,

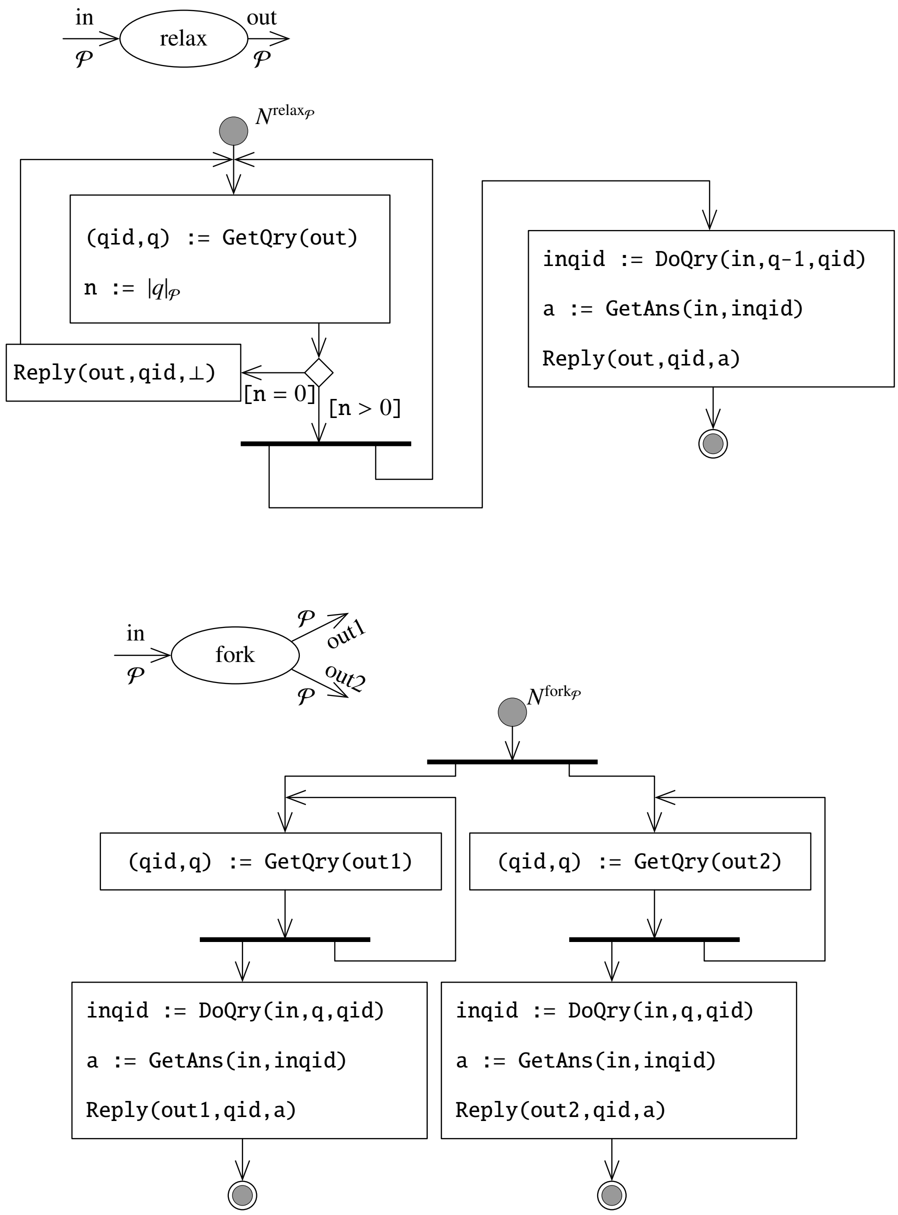

Figure 3: Definitions of processes $N^{\text{relax}_{\mathcal{P}}}$ and $N^{\text{fork}_{\mathcal{P}}}$.

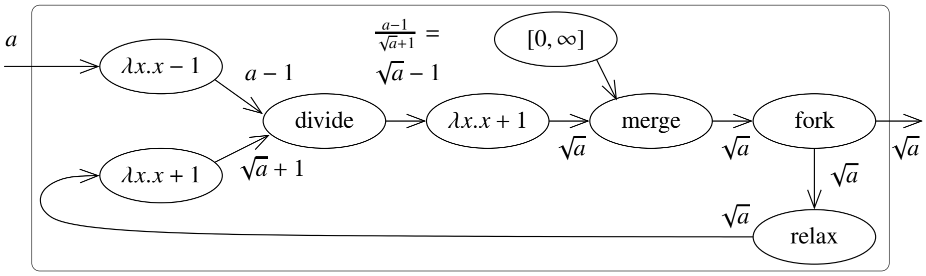

Figure 4: A network computing $\sqrt{a}$ for a fixed $a > 0$.

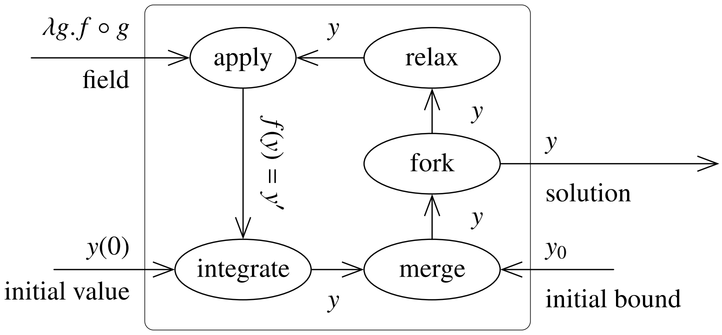

Figure 5: A network solving an initial value problem using the Picard operator

it then shows that the composition at the level of our more abstract semantics is safe with respect to the trace-level semantics. Safety has to be complemented by liveness and consistency analyses to rule out that safety is achieved though inactivity or explicit error output. Therefore, in Section 5, we provide one way to compositionally deduce liveness (which we call responsiveness in our context) using our "backwards" output-to-input query semantics. Section 6 comprises a proof that composition does not spoil our version of process consistency. Finally, in Section 7, we extend the composition results to infinite recursively defined networks.

2 Network structure and traces

Definition 1. A *process network* $\mathcal{N}$ consists of

– A countable set $P^{\mathcal{N}}$ of *process names*.

– A family of *processes* $\left(\mathrm{N}_p\right)_{p \in P^{\mathcal{N}}}$ indexed by process names.

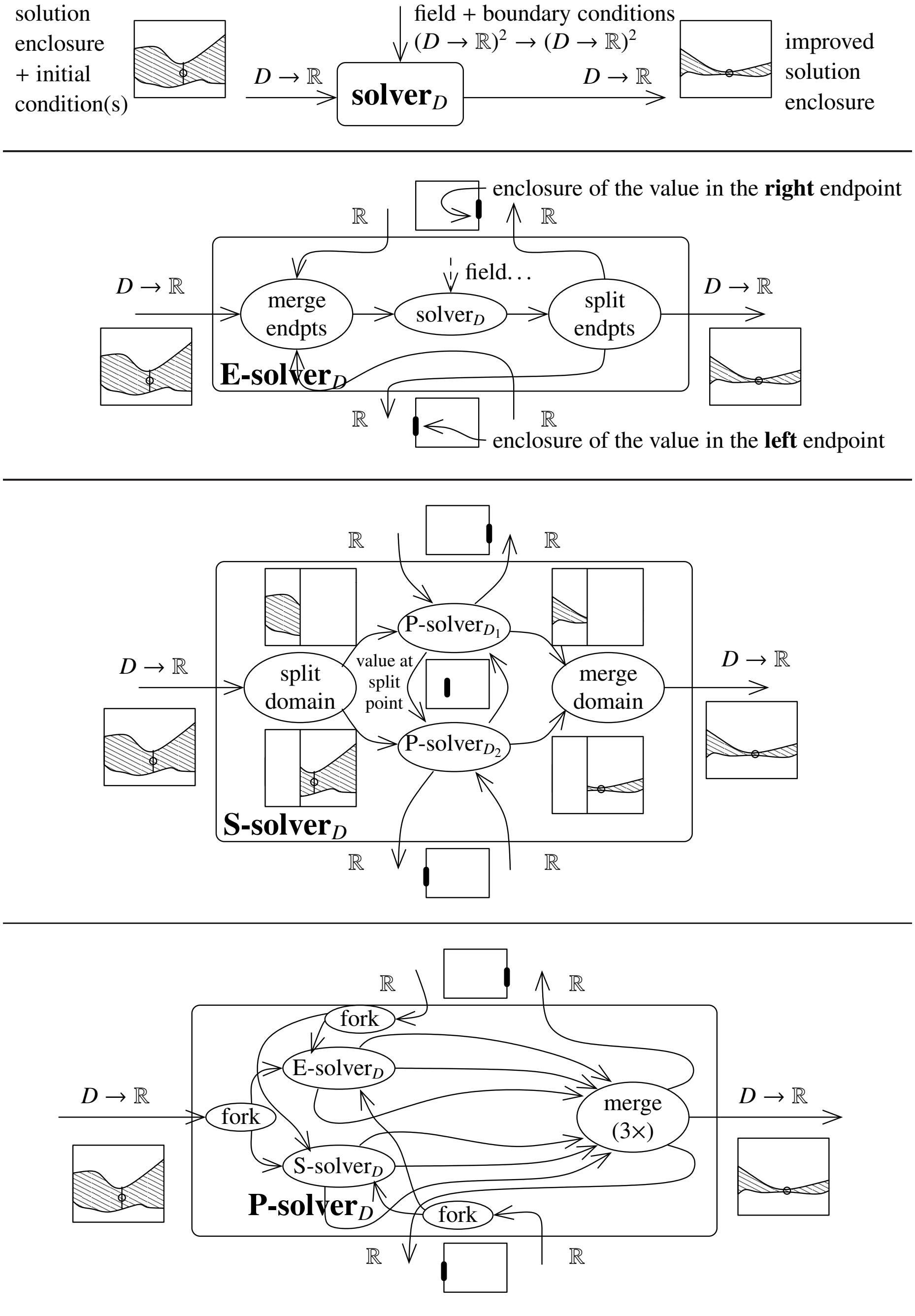

Figure 6: Parallelisation of an enclosure-based IVP/BVP solver for ODEs.

- A pairwise disjoint family of countable sets $(C^{\mathcal{N}}_{p_1,p_2})_{p_1,p_2\in P^{\mathcal{N}}}$ indexed by pairs of process names where each set $C^{\mathcal{N}}_{p_1,p_2}$ contains *names of channels* through which process N_{p_2} sends queries to and receives answers from process N_{p_1}. Moreover, let $C_{\mathcal{N}} := \bigcup_{p_1,p_2\in P^{\mathcal{N}}} C^{\mathcal{N}}_{p_1,p_2}$ and for a channel $c \in C^{\mathcal{N}}_{p_1,p_2} \subseteq C_{\mathcal{N}}$, define $N^+_c := \mathrm{N}_{p_1}$ and $N^-_c := \mathrm{N}_{p_2}$.
- A pair of functions $.^+$ and $.^-$ mapping a channel name c to process-labelled *sockets* c^+ and c^-:

$$(\forall p_1, p_2 \in P^{\mathcal{N}}, c \in C^{\mathcal{N}}_{p_1,p_2})\Big(c^+ = (p_1, s^+) \text{ and } c^- = (p_2, s^-)\Big)$$

  satisfying the following conditions:

  - $s^+ \in S^+_{\mathrm{N}_{p_1}}$ and $s^- \in S^-_{\mathrm{N}_{p_2}}$
  - The input and output sockets have matching protocols:

$$\Big(c^+ = (p_1, s^+), c^- = (p_2, s^-)\Big) \implies (\mathcal{P}^{s^+}_{\mathrm{N}_{p_1}} = \mathcal{P}^{s^-}_{\mathrm{N}_{p_2}})$$

    This common protocol is denoted $\mathcal{P}^c_{\mathcal{N}}$.

  - Each socket is connected to at most one channel, i. e. $.^+$ and $.^-$ are injections.

  The superscript $\mathcal{N}$ is dropped when no confusion is likely to arise. For further convenience we set $C^- := \{c^- \mid c \in C\}$ and $C^+ := \{c^+ \mid c \in C\}$.

- Disjoint sets of symbols $S^-_{\mathcal{N}}$ and $S^+_{\mathcal{N}}$ naming the network's *open input and output sockets*, respectively, using the following bijections:

$$\sigma^-\colon S^-_{\mathcal{N}} \to \{(p, s^-) \mid p \in P^{\mathcal{N}}, s^- \in S^-_{\mathrm{N}_p}, (p, s^-) \notin C^-\}$$
$$\sigma^+\colon S^+_{\mathcal{N}} \to \{(p, s^+) \mid p \in P^{\mathcal{N}}, s^+ \in S^+_{\mathrm{N}_p}, (p, s^+) \notin C^+\}$$

  We also set $S_{\mathcal{N}} := S^+_{\mathcal{N}} \cup S^-_{\mathcal{N}}$ and for each $s \in S_{\mathcal{N}}$ with $(p', s') = \sigma(s)$ where $\sigma = \sigma^+ \cup \sigma^-$, the socket's protocol $\mathcal{P}^{s'}_{\mathrm{N}_{p'}}$ is denoted either $\mathcal{P}^s_{\mathcal{N}}$ or $\mathcal{P}^{(p',s')}_{\mathcal{N}}$.

In the formation of a network, some sockets on some processes may be left attached to no channels — these are the *open* sockets introduced above. Later on (Def. 5) it will be shown how a network is encapsulated into a *composite process* with these sockets as its process sockets.

Let us now formally define the square root network introduced informally in Fig. 4, which we will denote $\mathcal{N}^{\sqrt{\cdot}}$:

- $P^{\mathcal{N}^{\sqrt{\cdot}}} := \{\text{dec}, \text{div}, \text{incA}, \text{nneg}, \text{mrg}, \text{frk}, \text{rlx}, \text{incB}\}$
- The assignment of processes to names is as shown in Fig. 7 where each process name is followed by a colon and its assigned process.

- The family of channel name sets consists of empty sets except for:
  - $C_{\mathrm{dec,div}} = \{\mathrm{c1}\}$ with $\mathrm{c1}^+ = (\mathrm{dec}, \mathtt{out})$, $\mathrm{c1}^- = (\mathrm{div}, \mathtt{num})$
  - and analogously for channels c2–c8 as indicated in Fig. 7.
- The input open socket is named $S^-_{\mathcal{N}^{\sqrt{\cdot}}} := \{\mathtt{in}\}$ and the output one $S^+_{\mathcal{N}^{\sqrt{\cdot}}} := \{\mathtt{out}\}$.

Also, we will consider the following slight modifications of $\mathcal{N}^{\sqrt{\cdot}}$:

- $\mathcal{N}^{\sqrt{\cdot}}_{\mathrm{mrg0}}$: The process mrg is modified so that it does not forward each query to both input sockets but forwards query 0 only to socket `in0` and all other queries only to socket `in`.
- $\mathcal{N}^{\sqrt{\cdot}}_{\mathrm{norlx}}$: Process rlx and channel c7 are removed, channel c6 is connected to process incB.

It should be intuitively clear that $\mathcal{N}^{\sqrt{\cdot}}_{\mathrm{mrg0}}$ behaves almost identically to $\mathcal{N}^{\sqrt{\cdot}}$, while $\mathcal{N}^{\sqrt{\cdot}}_{\mathrm{norlx}}$ does not answer any query larger than 0 due to unbounded looping in its cycle.

Definition 2 (The trace set of a network). For any network $\mathcal{N}$, we define the set of its traces, denoted $\mathfrak{T}_{\mathcal{N}}$, as the smallest set satisfying:

- Every $\zeta \in \mathfrak{T}_{\mathcal{N}}$ is an interleaving of its process-indexed projections $(\zeta_p)_{p \in P^{\mathcal{N}}}$ and each ζ_p is in $\mathfrak{T}_{\mathrm{N}_p}$.
- For each trace $\zeta = (\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathfrak{T}_{\mathcal{N}}$ we have:
  - The direct causality relation $\rhd\!\!\bullet$ is a *bijection* from the event subset $\gamma_{c^-}(E_{c^-}) \subset E$ onto the event subset $\gamma_{c^+}(E_{c^+}) \subset E$.
    (I. e. the channel c transfers events between the processes in a one-to-one manner.)

    In the above ζ_{c^+}, ζ_{c^-} are composite projections from ζ first to the processes and then to the sockets that the channel connects and $\gamma_{c^+}, \gamma_{c^-}$ are the composite event inclusion maps (see [Konečný and Farjudian 2010, Def. 3]) associated with these projections, respectively.
  - Moreover, whenever $\beta^- \rhd\!\!\bullet \beta^+$ for $\beta^- \in \gamma_{c^-}(E_{c^-})$ and $\beta^+ \in \gamma_{c^+}(E_{c^+})$, then $\eta(\beta^-) = \eta(\beta^+)$.
    (I. e. the channel c correctly transfers all queries and answers.)
  - For each event $\beta \in E$ if the set $\{\beta' \mid \beta \rhd \beta'\}$ contains no unanswered queries, then it is finite.
    (I. e. each answer is obtained in finite time.)

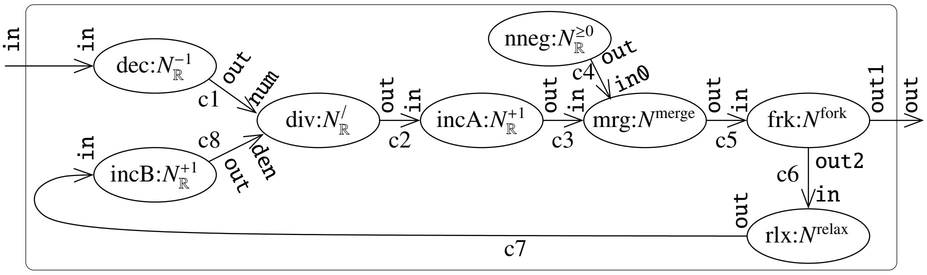

Figure 7: Network $\mathcal{N}^{\sqrt{\cdot}}$: process and channel assignments. All channels use the real number protocol with natural number queries and rational interval answers.

β	direct cause	$\eta(\beta)$	query-focused event description
0		(frk, (`out1`, 1 ?↦! [1, 2]))	external query registered by fork
1	0	(frk, (`in`, 1 ?↦! [1, 2]))	fork forwards the query to merger
2	1	(mrg, (`out`, 1 ?↦! [1, 2]))	merger registers the query
3	2	(mrg, (`in`, 1 ?↦! [1, 2]))	merger forwards the query to incrementer A
4	3	(incA, (`out`, 1 ?↦! [1, 2]))	incrementer A registers the query
5	4	(incA, (`in`, 1 ?↦! [0, 1]))	incrementer A forwards the query to divider
6	5	(div, (`out`, 1 ?↦! [0, 1]))	divider registers the query
7	6	(div, (`num`, 1 ?↦! [1, 1]))	divider forwards the query to decrementer
8	6	(div, (`den`, 1 ?↦! [1, ∞]))	divider forwards the query to incrementer B
9	7	(dec, (`out`, 1 ?↦! [1, 1]))	decrementer registers the query
10	8	(incB, (`out`, 1 ?↦! [1, ∞]))	incrementer B registers the query
11	10	(incB, (`in`, 1 ?↦! [0, ∞]))	incrementer B forwards the query to relaxer
12	9	(dec, (`in`, 1 ?↦! [2, 2]))	decrementer forwards the query outside
13	11	(rlx, (`out`, 1 ?↦! [0, ∞]))	relaxer registers the query
14	13	(rlx, (`in`, 0 ?↦! [0, ∞]))	relaxer forwards the relaxed query to fork
15	14	(frk, (`out2`, 0 ?↦! [0, ∞]))	relaxer's query registered by fork
16	15	(frk, (`in`, 0 ?↦! [0, ∞]))	fork forwards the query to merger
17	16	(mrg, (`out`, 0 ?↦! [0, ∞]))	merger registers the query
18	17	(mrg, (`in0`, 0 ?↦! [0, ∞]))	merger forwards the query to initialiser
19	18	(nneg, (`out`, 0 ?↦! [0, ∞]))	initialiser registers the query

Figure 8: An example trace of the network $\mathcal{N}^{\sqrt{\cdot}}$.

To illustrate the definition of the network trace set, let us peep inside the trace set of the square root network $\mathcal{N}^{\sqrt{\cdot}}$. A typical trace from this set would look as shown in Fig. 8. For the sake of readability, the events are listed by the likely time interleaving of the query components. This ordering is not part of the trace and one has to remember that answers occur in a different sequence than queries.

3 Network semantics

To simplify our reasoning about semantics, we will usually assume that all processes in a network are without hidden state.

Definition 3. A network is said to be *without hidden state* iff all its processes are such.

Definition 4 (Fixed point semantics of network).
For a network $\mathcal{N}$ without hidden state, we define its *fixed point QA-semantics* as follows:

$$\llbracket \mathcal{N} \rrbracket_{\mathrm{QA}} : \prod_{(p,s^-)\in\sigma(S^-_{\mathcal{N}})} \left\llbracket \mathcal{P}^{(p,s^-)}_{\mathcal{N}} \right\rrbracket_{\mathrm{QA}} \to \prod_{o\in C_{\mathcal{N}}\cup\sigma(S_{\mathcal{N}})} \left\llbracket \mathcal{P}^{o}_{\mathcal{N}} \right\rrbracket_{\mathrm{QA}}$$

$$\llbracket \mathcal{N} \rrbracket_{\mathrm{QA}} \left(\left(d_{(p,s^-)} \right)_{(p,s^-)\in\sigma(S^-_{\mathcal{N}})} \right) := \mathrm{LFP}\,(\Phi) = \bigsqcup_{\alpha \text{ ordinal}} \Phi^{\alpha}(\bot)$$

where Φ is a monotone endofunction defined in terms of the tuple $(d_{(p,s^-)})_{(p,s^-)\in\sigma(S^-_{\mathcal{N}})}$ as follows:

$$\left(\Phi\left(\left(d^{\mathrm{A}}_{o}\right)_{o\in C_{\mathcal{N}}\cup\sigma(S_{\mathcal{N}})}\right)\right)_{(p,s^-)} := d_{(p,s^-)}$$

(input sockets keep their initial semantics)

$$\left(\Phi\left(\left(d^{\mathrm{A}}_{o}\right)_{o\in C_{\mathcal{N}}\cup\sigma(S_{\mathcal{N}})}\right)\right)_{(p,s^+)} := \llbracket \mathrm{N}_p \rrbracket_{\mathrm{QA}} \left((d^{\mathrm{A}}_{c_{s^-}})_{s^-\in S^-_{\mathrm{N}_p}} \right)_{s^+}$$

(semantics of output socket given by its processes)

$$\left(\Phi\left(\left(d^{\mathrm{A}}_{o}\right)_{o\in C_{\mathcal{N}}\cup\sigma(S_{\mathcal{N}})}\right)\right)_{c} := \llbracket \mathrm{N}^+_c \rrbracket_{\mathrm{QA}} \left((d^{\mathrm{A}}_{c_{s^-}})_{s^-\in S^-_{\mathrm{N}^+_c}} \right)_{c^+}$$

(semantics of channel given by the outputting process)

where for each $s^- \in S^-_{\mathrm{N}_p}$, c_{s^-} is either a channel satisfying $(c_{s^-})^- = (p, s^-)$ or the input socket (p, s^-) if there is no such channel.

By reversing the roles of input and output sockets and replacing the QA-semantics with Q-semantics, we define the *fixed point Q-semantics* of $\mathcal{N}$ as follows:

$$\llbracket \mathcal{N} \rrbracket_{\mathrm{Q}} : \prod_{(p,s^+)\in\sigma(S^+_{\mathcal{N}})} \left\llbracket \mathcal{P}^{(p,s^+)}_{\mathcal{N}} \right\rrbracket_{\mathrm{Q}} \to \prod_{o\in C_{\mathcal{N}}\cup\sigma(S_{\mathcal{N}})} \left\llbracket \mathcal{P}^{o}_{\mathcal{N}} \right\rrbracket_{\mathrm{Q}}$$

$$\llbracket \mathcal{N} \rrbracket_{\mathrm{Q}} \left(\left(q_{(p,s^+)} \right)_{(p,s^+)\in\sigma(S^+_{\mathcal{N}})} \right) := \mathrm{LFP}\,(\Xi) = \bigsqcup_{\alpha \text{ ordinal}} \Xi^{\alpha}(\bot)$$

where $\varXi$ is a monotone endofunction defined in terms of the tuple $(q_{(p,s^+)})_{(p,s^+)\in\sigma(S^+_{\mathcal{N}})}$ as follows:

$$\left(\varXi\left(\left(q^{\mathrm{A}}_o\right)_{o\in C_{\mathcal{N}}\cup\sigma(S_{\mathcal{N}})}\right)\right)_{(p,s^+)} \coloneqq q_{(p,s^+)}$$

(output sockets keep their initial Q-semantics)

$$\left(\varXi\left(\left(q^{\mathrm{A}}_o\right)_{o\in C_{\mathcal{N}}\cup\sigma(S_{\mathcal{N}})}\right)\right)_{(p,s^-)} \coloneqq [\![\mathrm{N}_p]\!]_{\mathrm{Q}}\left((q^{\mathrm{A}}_{c_{s^+}})_{s^+\in S^+_{\mathrm{N}_p}}\right)_{s^-}$$

(Q-semantics of input socket given by its processes)

$$\left(\varXi\left(\left(q^{\mathrm{A}}_o\right)_{o\in C_{\mathcal{N}}\cup\sigma(S_{\mathcal{N}})}\right)\right)_{c} \coloneqq [\![\mathrm{N}^+_c]\!]_{\mathrm{Q}}\left((q^{\mathrm{A}}_{c_{s^+}})_{s^+\in S^+_{\mathrm{N}^-_c}}\right)_{c^-}$$

(Q-semantics of channel given by the inputting process)

Next, we work out $[\![\mathcal{N}^{\sqrt{\cdot}}]\!]_{\mathrm{QA}}$ as an example application of this definition. Let d^i denote the i-th iteration of applying the operator Φ to $\perp$ when computing the least fixed point. Note that i could be a transfinite ordinal when the processes in the network have discontinuous semantics.

Assume that the input socket's semantics is $d^1_{(\mathtt{in},\mathrm{dec})}(i) = a_i = [a^L_i, a^U_i]$ with $\bigcap_{i\in\omega} a_i = [a, a]$ for some $a \in \mathbb{R}, a > 1$. Also assume that $a^L_i \geq 1$ for all i. Note that on output sockets and on the channels d^1 has the initial assignment of semantic elements to a tuple of bottoms, i. e. d^1_s is $\lambda q.[-\infty, \infty]$.

After the second iteration, the value of d^2 on channel c1 already stabilises on $\lambda q.[a^L_q - 1, a^U_q - 1]$ and on channel c4 on $\lambda q.[0, \infty]$.

In the third iteration, the mapping on channel c5 "inherits" the mapping $\lambda q.[0, \infty]$ from c4 via process mrg. Consequently, in the fourth iteration, c6 and the output socket also get exactly the same improvement. In the fifth iteration, c7 gains a map with $q \mapsto [0, \infty]$ for $q > 0$ and afterwards c8 gains a map with $q \mapsto [1, \infty]$ for $q > 0$.

The most interesting evolution happens around process div. While the value on c1 remains static, on c8 and c2 it evolves as follows:

iteration	c8	c2
6, 7	$q \mapsto [1, \infty]$ for $q > 0$	$q \mapsto [0, a^U_q - 1]$ for $q > 0$
12, 13	$q \mapsto [2, a^U_{q-1} + 1]$ for $q > 1$	$q \mapsto [\frac{a^L_q - 1}{a^U_{q-1}+1}, \frac{a^U_q - 1}{2}]$ for $q > 1$
$\vdots$	$\vdots$	$\vdots$

After careful elementary analysis we observe that the least fixed point on the output socket is the following map:

$$\begin{aligned}
q &\mapsto \left[1 + \cfrac{a^L_q - 1}{2 + \cfrac{a^U_{q-1} - 1}{2 + \cfrac{a^L_{q-2} - 1}{\ddots (2 + (a^{U/L}_0 - 1))}}}, 1 + \cfrac{a^U_q - 1}{2 + \cfrac{a^L_{q-1} - 1}{2 + \cfrac{a^U_{q-2} - 1}{\ddots (2 + (a^{L/U}_0 - 1))}}} \right] \qquad \text{for } q > 0, \\
\bot, 0 &\mapsto [0, \infty]
\end{aligned}$$

where for convenience we override the names a^L_0 and a^L_1 so that $a^L_0 = 0$ and $a^L_1 = 1$. Let us denote the intervals in this output series $r_q = [r^L_q, r^U_q]$.

We do not include a long and fully general analysis of this sequence. Using elementary methods we have proved that $\sqrt{a} \in \bigcap_{q \in \omega} r_q$ for any representation of the input value a of the form specified earlier.

Convergence to a singleton is harder to prove, yet we believe that it holds in general. In the special case where the input a is represented exactly for all query indices (i. e. $a^L_q = a^U_q = a$ for all q), it is easy to show that

$$\frac{(r^L_{q+2})^2 - a}{(r^L_q)^2 - a} = \left(\frac{a - 1}{2r^L_q + a + 1} \right)^2 \qquad \text{and} \qquad \frac{(r^U_{q+2})^2 - a}{(r^U_q)^2 - a} = \left(\frac{a - 1}{2r^U_q + a + 1} \right)^2 \tag{1}$$

which means that if $a \in r^2_q$, then $a \in r^2_{q+2}$ and the width of r_q decreases exponentially with rate approximately $((\sqrt{a} - 1)/(\sqrt{a} + 1))^q$.

We would now hope that an analysis such as this one is sufficient to conclude that the network computes $\sqrt{a}$ when executed. It is not hard to verify that it is the case for $\mathcal{N}^{\sqrt{\cdot}}$ but it remains to be explained when it is the case in general. Theorem 7 below is one essential component of a general safety analysis.

4 Process composition

Definition 5 (Process composition). Each network $\mathcal{N}$ defines a new *composite process* $\mathrm{N}_{\mathcal{N}}$ whose:

- sockets are the network's open sockets, i. e.

$$S^+_{\mathrm{N}_{\mathcal{N}}} := S^+_{\mathcal{N}}, \quad S^-_{\mathrm{N}_{\mathcal{N}}} := S^-_{\mathcal{N}}$$

- socket protocol assignment follows that of the component processes of the network to which the sockets belong, i. e. $\mathcal{P}^{(p,s)}_{\mathrm{N}_{\mathcal{N}}} := \mathcal{P}^s_{\mathrm{N}_p}$

- trace set is $\mathfrak{T}_{\mathrm{N}_{\mathcal{N}}} := \{\zeta^{\pm} \mid \zeta \in \mathfrak{T}_{\mathcal{N}}\}$
  where $\zeta^{\pm}$ is the restriction of ζ to the index set $S^-_{\mathcal{N}} \cup S^+_{\mathcal{N}}$ followed by a rearrangement of the event indices using the mapping $(p,(s,e)) \mapsto (\sigma^{-1}(p,s),e)$.

The *composed QA- and Q-semantics* of $\mathrm{N}_{\mathcal{N}}$, denoted $[\![\mathcal{N}]\!]_{\mathrm{QA}} \upharpoonright_{S^+_{\mathcal{N}}}$ and $[\![\mathcal{N}]\!]_{\mathrm{Q}} \upharpoonright_{S^-_{\mathcal{N}}}$, respectively, are obtained by restricting $[\![\mathcal{N}]\!]_{\mathrm{QA}}$ and $[\![\mathcal{N}]\!]_{\mathrm{Q}}$ to the socket components, ignoring the channels, and then renaming the socket indices using σ^{-1}.

Lemma 6. *If $\mathcal{N}$ is without hidden state, the process $\mathrm{N}_{\mathcal{N}}$ is also without hidden state.*

Proof. Consider a trace ζ of $\mathrm{N}_{\mathcal{N}}$ and one of its causally-closed subtraces ζ'. The trace ζ is a restriction of a network trace $\zeta_{\mathcal{N}}$. Any direct causality in ζ has to be derived from transitive causality in $\zeta_{\mathcal{N}}$. Thus there is also a causally-closed subtrace $\zeta'_{\mathcal{N}}$ of $\zeta_{\mathcal{N}}$ whose restriction is equal to ζ'. Now the restrictions of $\zeta'_{\mathcal{N}}$ to individual processes in $\mathcal{N}$ are all valid process traces because these processes are all without hidden state. Thus $\zeta'_{\mathcal{N}}$ is a valid network trace and ζ' is a valid trace for $\mathrm{N}_{\mathcal{N}}$. □

The process $\mathrm{N}_{\mathcal{N}}$ also has its QA- and Q-semantics $[\![\mathrm{N}_{\mathcal{N}}]\!]_{\mathrm{QA}}$ and $[\![\mathrm{N}_{\mathcal{N}}]\!]_{\mathrm{Q}}$, respectively, derived using traces. These semantics reflect the execution of the network more closely but they are harder to describe in concrete instances than the composed semantics as one has to first describe the set of all network traces and derive the semantics from them. The following theorems show that our composed semantics safely approximate the more accurate trace-level semantics:

Theorem 7 (Safety of composed QA-semantics).
For any network $\mathcal{N}$ without hidden state we have $[\![\mathcal{N}]\!]_{\mathrm{QA}} \upharpoonright_{S^+_{\mathcal{N}}} \sqsubseteq [\![\mathrm{N}_{\mathcal{N}}]\!]_{\mathrm{QA}}$.

Theorem 8 (Safety of composed Q-semantics).
For any network $\mathcal{N}$ without hidden state we have $[\![\mathcal{N}]\!]_{\mathrm{Q}} \upharpoonright_{S^-_{\mathcal{N}}} \sqsupseteq [\![\mathrm{N}_{\mathcal{N}}]\!]_{\mathrm{Q}}$.

Proof. (Theorem 7) We use the trace-based definition of QA-semantics from [Konečný and Farjudian 2010, Def. 23]. Pick an arbitrary trace $\zeta \in \mathfrak{T}_{\mathcal{N}}$. To prove the inequality, we show that

$$[\![\mathcal{N}]\!]_{\mathrm{QA}} \upharpoonright_{S^+_{\mathcal{N}}} \left(\left(\left[\!\left[\zeta_{(p,s^-)}\right]\!\right]_{\mathrm{QA}} \right)_{(p,s^-)\in S^-_{\mathcal{N}}} \right) \sqsubseteq \left(\left[\!\left[\zeta_{(p,s^+)}\right]\!\right]_{\mathrm{QA}} \right)_{(p,s^+)\in S^+_{\mathcal{N}}}$$

i. e. the composed semantics safely estimates any actual execution of the network. We prove this by transfinite induction over the number of iterations used to compute the least fixed point. In fact, we strengthen the claim so that it not only includes the output sockets of the network but also the channels and input sockets, i. e. for all $c \in C_{\mathcal{N}}$, $(p,s) \in \sigma(S_{\mathcal{N}})$ and (ordinal numbers) $i \geq 0$:

$$\left(\Phi^i(\bot)\right)_c \sqsubseteq [\![\zeta_{c^+}]\!]_{\mathrm{QA}} \quad \text{and} \quad \left(\Phi^i(\bot)\right)_{(p,s)} \sqsubseteq \left[\!\left[\zeta_{(p,s)}\right]\!\right]_{\mathrm{QA}} \qquad (2)$$

This is clearly true for $i = 0$ because the components of $\Phi^0(\bot)$ for channels and output sockets are bottom. Assuming that (2) holds for i, we prove it holds for $i+1$ as well. Using the definition of $\Phi^{i+1}(\bot)$ for a channel c we get:

$$\left(\Phi^{i+1}(\bot)\right)_c = [\![\mathrm{N}_c^+]\!]_{\mathrm{QA}}\left(\left(\left(\Phi^i(\bot)\right)_{c_{s^-}}\right)_{s^-\in S^-_{\mathrm{N}_c^+}}\right)_{c^+} \sqsubseteq [\![\mathrm{N}_c^+]\!]_{\mathrm{QA}}\left(\left([\![\zeta_{c_{s^-}}]\!]_{\mathrm{QA}}\right)_{s^-\in S^-_{\mathrm{N}_c^+}}\right)_{c^+}$$

by the induction hypothesis and monotonicity of the QA-semantics for the process N_c^+. The definition of process QA-semantics also yields that the right-hand-side value is the *infimum* over all process traces that are compatible with the given values on the input sockets. Thus for such a particular trace, namely $\zeta_{\mathrm{N}_c^+}$, the value has to be below the semantics of this trace:

$$\left(\Phi^{i+1}(\bot)\right)_c \sqsubseteq [\![\mathrm{N}_c^+]\!]_{\mathrm{QA}}\left(\left([\![\zeta_{c_{s^-}}]\!]_{\mathrm{QA}}\right)_{s^-\in S^-_{\mathrm{N}_c^+}}\right)_{c^+} \sqsubseteq [\![\zeta_{c^+}]\!]_{\mathrm{QA}}$$

The same inequality is proved similarly for output sockets instead of channels and for input sockets it is trivial.

Now assume that α is a limit ordinal and (2) holds for all $i < \alpha$. We have

$$\begin{cases} (\Phi^\alpha(\bot))_c &= \bigsqcup\{(\Phi^i(\bot))_c \mid i < \alpha\} \\ (\Phi^\alpha(\bot))_{(p,s^+)} &= \bigsqcup\{(\Phi^i(\bot))_{(p,s^+)} \mid i < \alpha\} \end{cases}$$

which implies:

$$\begin{cases} (\Phi^\alpha(\bot))_c &\sqsubseteq [\![\zeta_{c^+}]\!]_{\mathrm{QA}} \\ (\Phi^\alpha(\bot))_{(p,s^+)} &\sqsubseteq \left[\!\left[\zeta_{(p,s^+)}\right]\!\right]_{\mathrm{QA}} \end{cases}$$

We have proved that all approximations of the least fixed point satisfy (2). By the completeness of the semantic lattices, we get that the least fixed point too — as the supremum of all these approximations — satisfies (2). □

Proof. (Theorem 8) To prove the inequality $[\![\mathcal{N}]\!]_{\mathrm{Q}} \upharpoonright_{S^-_{\mathcal{N}}} \sqsupseteq [\![\mathrm{N}_{\mathcal{N}}]\!]_{\mathrm{Q}}$, as in the statement of the theorem, recall from [Konečný and Farjudian 2010, Def. 18] that the right-hand-side is a supremum of tuples $([\![\zeta_{s^-}]\!]_{\mathrm{Q}})_{s^-\in S^-_{\mathcal{N}}}$ over all network traces ζ in which the events used to define this tuple are all caused by the events that define the tuple $([\![\zeta_{s^+}]\!]_{\mathrm{Q}})_{s^+\in S^+_{\mathcal{N}}}$. It suffices to show that this tuple of query elements is below or equal to $[\![\mathcal{N}]\!]_{\mathrm{Q}} \upharpoonright_{S^-_{\mathcal{N}}}$, i.e.

$$([\![\zeta_{s^-}]\!]_{\mathrm{Q}})_{s^-\in S^-_{\mathcal{N}}} \sqsubseteq [\![\mathcal{N}]\!]_{\mathrm{Q}} \upharpoonright_{S^-_{\mathcal{N}}}\left(([\![\zeta_{s^+}]\!]_{\mathrm{Q}})_{s^+\in S^+_{\mathcal{N}}}\right) \tag{3}$$

for each trace $\zeta \in \mathfrak{T}_{\mathcal{N}}$. We argue that the inequality (3) follows if

$$q_k \sqsubseteq [\![\mathcal{N}]\!]_{\mathrm{Q}} \upharpoonright_{S^-_{\mathcal{N}}}\left((\bot)_{s^+\in S^+_{\mathcal{N}}}[\sigma^{-1}(p_0, s_0) \mapsto q_0]\right) \tag{4}$$

is true for *each* chain of directly caused events

$$\left(p_0, (s_0, (q_0, x_0))\right) \rhd\!\!\bullet \left(p_1, (s_1, (q_1, x_1))\right) \rhd\!\!\bullet \ldots \rhd\!\!\bullet \left(p_k, (s_k, (q_k, x_k))\right) \tag{5}$$

leading from an output socket $(p_0, s_0) \in \sigma(S^+_{\mathcal{N}})$ to an input socket $(p_k, s_k) \in \sigma(S^-_{\mathcal{N}})$, appearing in any $\zeta \in \mathfrak{I}_{\mathcal{N}}$.

Statement (4) implies (3) because both sides of (3) are the supremum of the appropriate sides of (4) indexed by an appropriate set of chains as in (5).

To prove (4), recall $[\![\mathcal{N}]\!]_{\mathrm{Q}} \restriction_{S^-_{\mathcal{N}}}$ is the least fixed point, which can be expressed by iterating an update operator on a mapping from all channels and sockets to query elements, starting with the constant bottom mapping except for the output channel (p_0, s_0) which gets q_0. We will show that the k-th iteration of this update operator — and by implication the least fixed point — is already larger than q_k, thus proving (4).

To see that the k'th iteration is larger than q_k, recall that the update operator uses the Q-semantics of all processes each raising a typical q_i to at least q_{i+1} for the channel or socket associated with socket (p_{i+1}, s_{i+1}) involved in the causality:

$$\big(p_i, (s_i, (q_i, x_i))\big) \rhd\!\!\bullet \big(p_{i+1}, (s_{i+1}, (q_{i+1}, x_{i+1}))\big)$$

□

5 Network responsiveness

As discussed in [Konečný and Farjudian 2010, Sect. 5.3.3], even when we know that a process' QA-semantics is exactly what we want, it does not guarantee desirable execution in each instance. A lower bound on the QA-semantics of a process such as the one provided by Theorem 7, if not analysed correctly could be more misleading than informative as the true QA-semantics may be equal to $\top$ for some queries, meaning that these queries and those above are never answered or are always answered inconsistently. Fortunately, when we add responsiveness and QA-consistency for a network to a good lower bound on its QA-semantics, these properties together guarantee correct execution. It is thus important to research compositionally manageable conditions under which networks certainly preserve responsiveness and also to find a way to prove that a responsive network behaves consistently among traces. In this section we address responsiveness and in the following one consistency.

An example network that does not preserve responsiveness is $\mathcal{N}^{\sqrt{\cdot}}_{\mathrm{norlx}}$. Recall that $\mathcal{N}^{\sqrt{\cdot}}_{\mathrm{norlx}}$ differs from $\mathcal{N}^{\sqrt{\cdot}}$ in that it has no rlx process. When computing the fixed-point QA-semantics by iterating Φ, all updates in the d^i mappings occur at index 0 and the fixed point is reached in finitely many iterations. Focusing on the div process we see the following:

iteration	c8	c2
$5, 6$	$0 \mapsto [1, \infty]$	$0 \mapsto [0, a^U_0 - 1]$
$10, 11$	$0 \mapsto [1, \infty]$	$0 \mapsto [0, a^U_0 - 1]$
$\vdots$	$\vdots$	$\vdots$

The least fixed point on the output socket will be a mapping that assigns $0 \mapsto [0, \infty]$ and all other indices remain assigned with bottom.

Here the QA-semantics reveals that something goes wrong because it is rather uninformative. Nevertheless, it does not reveal the fact that some queries will never be answered. E.g. when sending the query 1, it will propagate through the cycle in the network round and round forever without receiving or providing any answers, resulting in a *livelock*.

A worse case is a variant of the $\mathcal{N}^{\sqrt{\cdot}}$ network in which the rlx process is faulty and it forwards each query twice — once reduced, as it should, and once not reduced. The faulty rlx does not answer its query until both its induced queries are answered. When both are answered, it forwards the answer to the reduced query and ignores the other one. The semantics of the faulty rlx process does not change at all and thus also the semantics of the network remains the same, indicating convergence towards $\sqrt{a}$. Nevertheless, the network does not answer any query except 0.

To obtain necessary conditions for preserving responsiveness, we analyse cycles, using the following definition:

Definition 9 (Removing network channels). For a network $\mathcal{N}$ and a subset $C' \subseteq C_{\mathcal{N}}$ of its channels, let $\mathcal{N}[C']$ denote the network obtained from $\mathcal{N}$ by removing the channels C' and for each $c \in C'$ naming its two new sockets c^{in} and c^{out} with $\sigma(c^{\text{in}}) = c^-$ and $\sigma(c^{\text{out}}) = c^+$.

We will present a sufficient condition guaranteeing that a network preserves responsiveness, based on Q-semantics and the same measure of progress we used to define convergence rate:

Definition 10. A network $\mathcal{N}$ is called *Q-decreasing* iff there exists a subset of channels $C^{\oslash} \subseteq C$ for which $\mathcal{N}[C^{\oslash}]$ contains no loops and its Q-semantics $[\![\mathcal{N}[C^{\oslash}]]\!]_{\mathrm{Q}}$ is decreasing in the following sense:

$$\forall h : \mathbb{N} \to C^{\oslash}, \forall q_0 \in [\![\mathcal{P}_{\mathcal{N}}^{h(0)}]\!]_{\mathrm{Q}} : \quad |q_0| > 0 \Rightarrow \exists k \in \mathbb{N} : |q_k| < |q_0|$$

in which for every $i \in \mathbb{N}$

$$q_{i+1} := \left(\left([\![\mathcal{N}[C^{\oslash}]]\!]_{\mathrm{Q}}^{h(i)^{\text{out}}}\right)(q_i)\right)_{h(i+1)^{\text{in}}}$$

In other words, when querying $\mathcal{N}[C^{\oslash}]$ with q_0 on socket $h(0)^{\text{out}}$, observing what consequent queries there are on $h(1)^{\text{in}}$, forwarding those queries over to $h(1)^{\text{out}}$ and observing what consequent queries there are on $h(2)^{\text{in}}$ etc., one of the observed queries has strictly lower measure than q_0.

Notice that if some of the channels $c \in C^{\oslash}$ do not lie on any cycle (e. g. channels c1 and c4 in $\mathcal{N}^{\sqrt{\cdot}}$, see Fig. 7), the channel split causes the network to be divided into several independent parts. In particular, sending the query q_c on socket c^{out} has no influence on what happens on socket c^{in} and other sockets in the other part. In this situation, if the sequence h crosses between two independent parts, the Q-semantics becomes $\perp$ and the inequality holds trivially.

The network $\mathcal{N}^{\sqrt{\cdot}}$ is Q-decreasing. For instance, in $\mathcal{N}^{\sqrt{\cdot}}[\{\text{c2}\}]$ any query on c2^{out} will have to pass via process rlx and thus if it has any consequence on c2^{in}, it will be smaller by 1. By the same argument the network $\mathcal{N}^{\sqrt{\cdot}}_{\text{mrg0}}$ is Q-decreasing as well.

On the other hand, the network $\mathcal{N}^{\sqrt{\cdot}}_{\text{norlx}}$ is not Q-decreasing, which we argue as follows. Each set $C^{\oslash}$ for $\mathcal{N}^{\sqrt{\cdot}}_{\text{norlx}}$ will have to contain at least one of the channels that appear on the cycle, i. e. c2, c3, c5, c6 or c8. (Recall that there is no c7 in $\mathcal{N}^{\sqrt{\cdot}}_{\text{norlx}}$.) Let $c \in C^{\oslash}$ be one of these channels. Then the sequence $h(i) = c$ and query $q_0 = 1$ gives $q_k = 1$ for all k, which means that the network cannot be Q-decreasing.

Theorem 11 (Composition preserves responsiveness). *If in a Q-decreasing network $\mathcal{N}$ all processes are responsive, the composed process $\text{N}_{\mathcal{N}}$ is also responsive.*

Proof. Assume that all premises are true but the process $\text{N}_{\mathcal{N}}$ is not responsive. This should yield a contradiction.

To break responsiveness, there has to be a network trace $\zeta = (\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathfrak{T}_{\mathcal{N}}$, whose restriction $\zeta' := \zeta\restriction_{S_{\mathcal{N}}}$, denoting $\zeta' = (\Sigma, E', \eta', \rhd\!\!\bullet')$, has an unanswered query $\beta_0 \in E' \subseteq E$ on an output socket and all queries on input sockets directly (with respect to $\rhd\!\!\bullet'$) caused by β_0 are answered. Assume that $\eta(\beta_0) = ((s_0^+, p_0^+), (q, \Omega))$.

Since the process $\text{N}_{p_0^+}$ is responsive, there has to be an unanswered query on its input socket that is a direct (with respect to $\rhd\!\!\bullet$) consequence of β_0. This query cannot lead to an external socket because we assume that all external queries directly (with respect to $\rhd\!\!\bullet'$) caused by β_0 are answered. Thus the query leads to a channel and has to be registered by the process on the other side of the channel. Let this event be β_1. Due to the definition of causality in networks, we have $\beta_0 \rhd \beta_1$.

By repeating this argument, we obtain an infinite sequence $\beta_0, \beta_1, \beta_2, \ldots$ of unanswered queries on output sockets $s_0, s_1, s_2, \ldots$ of processes $p_0, p_1, p_2, \ldots$ with $\beta_0 \rhd \beta_1 \rhd \beta_2 \rhd \ldots$.

Let $C^{\oslash}$ be a set of channels that provide evidence that the network is Q-decreasing. Since without these channels the network has no cycles, there have to be infinitely many indices $j_1 < j_2 < j_3 < \ldots$ with $\eta(\beta_{j_i}) = (c_i^+, (q_i, \Omega))$, $c_i \in C^{\oslash}$, i. e. the unanswered queries $q_1, q_2, \ldots$ happen all on some channels in $C^{\oslash}$, and each $\beta_{j_{i+1}}$ is caused by β_{j_i}.[2]

The trace ζ can be transformed by simply renaming indices to a legitimate trace of the split network $\mathcal{N}[C^{\oslash}]$, in particular retaining all the events $\beta_0, \beta_1, \beta_2, \ldots$ and all their properties. Now Def. 10 may be applied with $h(i) = c_i$, from which it follows that for each i either $\left|[\![q_i]\!]_{\text{Q}}\right| = 0$ or there is some $i' > i$ with $\left|[\![q_i]\!]_{\text{Q}}\right| > \left|[\![q_{i'}]\!]_{\text{Q}}\right|$. The former is impossible because only $\perp$ has $|\perp| = 0$ and no actual query can mean $\perp$. The latter is impossible because no infinite sequence of natural numbers can be strictly decreasing.

□

[2] The channels c_i, ($i \in \mathbb{N}$), do not need to — in fact *cannot* — be distinct.

The theorem can be strengthened by using a weaker Q-decreasing property — one relative to a certain input QA-semantics, using input relative Q-semantics as introduced in [Konečný and Farjudian 2010, Sect. 5.3.4]. To prove responsiveness under all circumstances, one has to quantify this property over all possible input QA-semantics.

6 Network consistency

In order to show that a network such as our example square root network is consistent, we prove that a responsive composition of QA-consistent processes is QA-consistent and at the same time strengthen Theorem 7 to give a semantic equality for such networks.

Theorem 12 (Composition of QA-consistent processes). *Let $\mathcal{N}$ be a responsive network of responsive QA-consistent processes without hidden state. Then* $\mathrm{N}_{\mathcal{N}}$ *is QA-consistent and* $[\![\mathcal{N}]\!]_{\mathrm{QA}} \restriction_{S^+_{\mathcal{N}}} = [\![\mathrm{N}_{\mathcal{N}}]\!]_{\mathrm{QA}}$.

Proof. Consider a trace $\zeta = (\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathfrak{T}_{\mathcal{N}}$ that has no unanswered queries and pick some event $\beta \in E$. Denote the query and answer in $\eta(\beta)$ as q and a, respectively. Since all events caused by β are answered, there are only finitely many of them. Using induction on the anti-reflexive relation $\rhd\!\!\bullet$ we show that for each β' with $\beta \rhd \beta'$ there exists $n \in \mathbb{N}$ with $\Phi^n([\![\zeta_o]\!]_{\mathrm{QA}})_{c'}(q') \sqsupseteq a'$ where $\eta(\beta') = (c', (q', a'))$ and Φ is the operator used in Def. 4. For events that cause no other events, this property holds for $n = 1$ thanks to the QA-consistency and lack of hidden state in the answering process. For other events, we set n to be one higher than the highest n for all events directly caused by this event. Again, this is correct thanks to the QA-consistency of and lack of hidden state in the answering process.

We have just proved that all actual answers in ζ are reached by some finite iteration of the Φ operator. It is therefore the case that the least fixed point of Φ is not below any of the traces' QA-semantics. In combination with Theorem 7, we get the desired equality and moreover, since the equality holds for every trace, also QA-consistency.

□

7 Infinite networks

The goal of this section is to formalise a method for specifying recursive process networks such as those illustrated in Figs 2 and 6 and to extend the concepts and results of the previous two subsections to the composite processes defined by such networks.

7.1 Recursive network families

Definition 13 (Dummy processes). A process is called *dummy* if all its traces consist purely of queries on its output sockets answered with $\top$.

Definition 14. A *recursive network family* is a set of tuples $(\mathcal{N}_i, P_i, \pi_i)_{i\in I}$ where I is a finite set and for each $i \in I$:

- $\mathcal{N}_i$ is a network
- $P_i \subseteq P^{\mathcal{N}_i}$ and π_i: $P_i \to I$ is a map such that for each $p \in P_i$:
  - the process N_p is dummy
  - the processes N_p and $\mathrm{N}_{\mathcal{N}_{\pi_i(p)}}$ have the same sockets and socket protocols.

Informally, each such dummy process N_p is a place holder within the network $\mathcal{N}_i$ for the process $\mathrm{N}_{\mathcal{N}_{\pi_i(p)}}$. The following definition builds on this intuition:

Definition 15 (Unfolding of recursive network families). The unfolding of a recursive network family $(\mathcal{N}_i, P_i, \pi_i)_{i\in I}$ is the sequence $(\mathcal{N}_i^0)_{i\in I}$, $(\mathcal{N}_i^1)_{i\in I}$, $(\mathcal{N}_i^2)_{i\in I}$, ... defined as:

- $\mathcal{N}_i^0 := \mathcal{N}_i$
- $\mathcal{N}_i^{n+1} := \mathcal{N}_i[\mathrm{N}_p \mapsto \mathrm{N}_{\mathcal{N}^n_{\pi_i(p)}}]$ which stands for the network $\mathcal{N}_i$ in which all processes $\mathcal{N}_p$ with $p \in P_i$ have been substituted by the indicated processes with compatible sockets.

To avoid introducing the same notation many times over, for the remainder of Section 7 assume $\mathcal{F} = (\mathcal{N}_i, P_i, \pi_i)_{i\in I}$ is some recursively defined family and $(\mathcal{N}_i^0)_{i\in I}$, $(\mathcal{N}_i^1)_{i\in I}$, $(\mathcal{N}_i^2)_{i\in I}$, ... is its unfolding.

As a concrete example, we let $\mathcal{F}^{\exp}$ denote the family $1 \mapsto (\mathcal{N}^{\mathrm{R}}, \{\mathrm{R}\}, \mathrm{R} \mapsto 1), 2 \mapsto (\mathcal{N}^{\exp}, \{\mathrm{R}\}, \mathrm{R} \mapsto 1)$ partially specified in Fig. 9, which corresponds to the infinite exponentiating network in Fig. 2. Assume that N^T is a modification of T_k from Fig. 2 abstracting k via a new natural number socket using the simplest protocol for natural numbers, which allows only traces with exactly one event that specifies the whole number as its answer. While technically not accurate, it helps one's intuition to replace both N^{dummy} in Fig. 9 with $\mathrm{N}_{\mathcal{N}^{\mathrm{R}}}\restriction_{\{\mathtt{x},\mathtt{k},\mathtt{out}\}}$.

7.2 Trace, process and network refinement

Our goal is to extend all our theorems about networks to recursive network families and processes defined by them. Most will follow naturally from the network theorems once we show that the semantics of the networks in the unfolding of a recursive family form a monotone sequence. To that end, we formalise a notion of trace and process refinement and show that unfolding forms a refinement sequence and all our semantics are monotone with respect to refinement.

Definition 16 (Trace refinement). A query-answer trace $\zeta_1 = (\Sigma, E_1, \eta_1, \rhd\!\!\bullet_1)$ is *refined* by another query-answer trace $\zeta_2 = (\Sigma, E_2, \eta_2, \rhd\!\!\bullet_2)$, written $\zeta_1 \sqsubseteq \zeta_2$, if and only if there is an injective map $\tau : E_1 \to E_2$ such that:

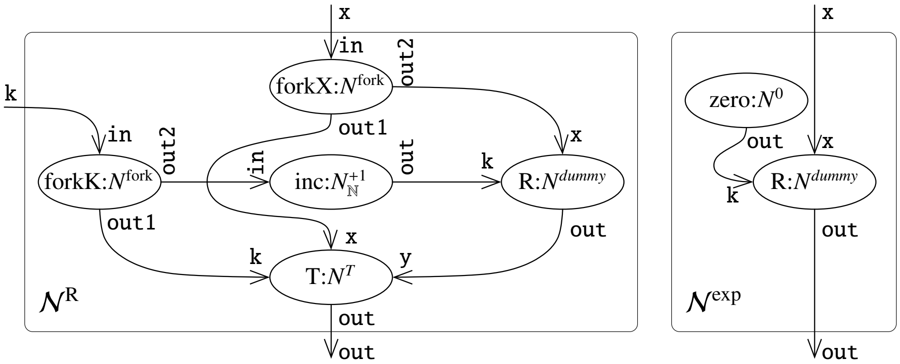

Figure 9: A recursive network family for an exponentiation infinite network.

- for any $\beta, \beta' \in E_1$ we have $\beta \rhd\!\!\bullet_1 \beta'$ if and only if $\tau(\beta) \rhd\!\!\bullet_2 \tau(\beta')$ (i. e. refinement keeps causality among existing events; τ is an order morphism);
- whenever $\eta_1(\beta) = (q_1, x_1)$ and $\eta_2(\tau(\beta)) = (q_2, x_2)$, then $q_1 = q_2$ and either $x_1 = x_2$ or $x_1 = \top$ (i. e. refinement does not change existing events except improving $\top$ answers);
- whenever $\beta \rhd\!\!\bullet_2 \beta'$ and β is outside the image of τ, then so is β' (i. e. events added by refinement cannot cause existing events);
- whenever $\beta' \in E_2$ is either outside the image of τ or its answer differs from its τ preimage, then for any $\beta \rhd\!\!\bullet_2 \beta'$ the event β is either also outside the image of τ or the answer in its τ preimage is $\top$ (i. e. refinement can add a consequence to existing event only if its answer is $\top$).

Whenever there is such a τ which is also a bijection, then we say ζ_1 *is closely refined by* ζ_2, denoted $\zeta_1 \sqsubseteq^{\bullet} \zeta_2$.

Refinement is also naturally extended to process and network traces — the definition for these traces changes only in the second point where the two events must agree in all components, including socket and process indices, except the answers.

Definition 17 (Process and network refinement). A process N_1 is refined by process N_2, written $\mathrm{N}_1 \sqsubseteq \mathrm{N}_2$, if and only if the two processes have the same sockets and socket protocol assignments and there is a surjection $r\colon \mathfrak{T}_{\mathrm{N}_2} \to \mathfrak{T}_{\mathrm{N}_1}$ with $r(\zeta) \sqsubseteq \zeta$ for all traces ζ and $r(\zeta)_{s^+} \sqsubseteq^{\bullet} \zeta_{s^+}$ for all output sockets s^+.

The definition of network refinement is word-for-word analogous to the definition of process refinement.

Note that structural similarity between a pair of refinement-related networks is implied by the refinement between the traces of the two networks.

Lemma 18 (Unfolding forms refinement chains). *The unfolding of a recursive network family satisfies $\mathcal{N}_i^n \sqsubseteq \mathcal{N}_i^{n+1}$ for each $i \in I, n \in \mathbb{N}$.*

Proof. Firstly, we have $\mathcal{N}_i^0 \sqsubseteq \mathcal{N}_i^1$ because a dummy process is trivially refined by any other process with the same sockets and protocols and substituting a process with its refinement in a network is a network refinement.

By induction on n we get the required property using the same argument. □

Lemma 19 (Semantics are monotone with respect to refinement).

1. *For a trace refinement $\zeta_1 \sqsubseteq \zeta_2$, we have* $[\![\zeta_1]\!]_{\mathrm{Q}} \sqsubseteq [\![\zeta_2]\!]_{\mathrm{Q}}$ *and* $[\![\zeta_1]\!]_{\mathrm{QA}} \sqsupseteq [\![\zeta_2]\!]_{\mathrm{QA}}$.
2. *For processes* $\mathrm{N}_1 \sqsubseteq \mathrm{N}_2$ *without hidden state, we have* $[\![\mathrm{N}_1]\!]_{\mathrm{Q}}^{\mathbf{T}} \sqsubseteq [\![\mathrm{N}_2]\!]_{\mathrm{Q}}^{\mathbf{T}}$ *and* $[\![\mathrm{N}_1]\!]_{\mathrm{QA}} \sqsupseteq [\![\mathrm{N}_2]\!]_{\mathrm{QA}}$.
3. *For networks $\mathcal{N}_1 \sqsubseteq \mathcal{N}_2$ without hidden state, we have* $[\![\mathcal{N}_1]\!]_{\mathrm{Q}} \sqsubseteq [\![\mathcal{N}_2]\!]_{\mathrm{Q}}$ *and* $[\![\mathcal{N}_1]\!]_{\mathrm{QA}} \sqsupseteq [\![\mathcal{N}_2]\!]_{\mathrm{QA}}$.

Proof. 1. ζ_1 contains fewer queries than ζ_2 and Q-semantics, as supremum of all queries, increases with more queries.

For any query element, we have a subset of relevant answers in ζ_1 compared to ζ_2 and possibly an extra $\top$ for ζ_1. Thus the QA-semantics, as the infimum of these answers, is decreasing.

2. Each trace in N_2 has a counterpart trace in N_1 whose Q-semantics is smaller on each input socket but same on each output socket. Thus the Q-semantics of N_2 is above the Q-semantics of N_1.

   Take some lower bound on the QA-semantics on the input sockets and all N_2 traces whose restrictions on the input sockets have QA-semantics above or equal this lower bound. The mapping r that witnesses the refinement translates these traces to N_1 traces that are still above the lower bound because their QA-semantics may have been only increased. There can be additional traces below the lower bound in N_1 thanks to some events on the input sockets disappearing or changing their answers to $\top$. We show that such traces do not influence the infimum of answers on output sockets and thus conclude the proof that the QA-semantics in N_1 is above that in N_2.

   To show that these “elevated” traces do not influence the infimum of answers on output sockets, we use the lack of hidden state to allow us to consider only traces in which all events are causally linked. The only way an answer in the QA-semantics on input socket gets elevated is via absence of some query or an answer becoming $\top$. In both cases all causing events either also disappear or get the answer $\top$, which includes all the events on output sockets. Thus their QA-semantics is $\top$, which means they are irrelevant when calculating QA-semantics.

3. The semantics of networks are derived using a least fixed point over the combined action of the semantics of all processes in the network on the channels and output sockets. If all process semantics decrease (increase), then the fixed point decreases (increases). □

7.3 Recursively defined networks and processes

Definition 20 (Recursively defined network and process). A *recursively defined network* is given by a recursive network family $\mathcal{F}$ and an index $u \in I$.

The trace set of such a network, is defined as:

$$\mathfrak{T}_{\mathcal{F},u} := \Big\{\zeta = (\Sigma, E, \eta, \rhd\!\bullet) \Big| \\ \forall E' \subseteq E \text{ finite and closed under } \rhd\!\bullet^{-1}\colon \exists n \in \mathbb{N} : \forall m > n : \zeta\!\restriction_{E'} \in \mathfrak{T}_{\mathcal{N}_u^m}\Big\}$$

A *recursively defined process* $\mathrm{N}_{\mathcal{F},u}$ is defined from a recursively defined network $(\mathcal{F}, u)$ by hiding its internal structure analogously to how a composed process is defined from a network.

For example, the exponentiation process defined by the infinite network illustrated in Fig. 9 is specified formally as $(\mathcal{F}^{\exp}, 2)$.

An essential property of a recursively defined network is that no query chain descends infinitely deep into it. The following definition captures this property formally.

Definition 21 (Finitely unfolding network). A recursively defined network $(\mathcal{F}, u)$ is called *finitely unfolding* if for each trace $\zeta = (\Sigma, E, \eta, \rhd\!\bullet)$ of this network and for each event $\beta \in E$, the subtrace generated from β by $\rhd\!\bullet$ is a valid trace in $\mathcal{N}_u^n$ for some $n \in \mathbb{N}$.

Theorem 22 (Responsiveness of recursively defined process). *Assume that the recursively defined network $(\mathcal{F}, u)$ is finitely unfolding and that for all $n \in \mathbb{N}$ and $i \in I$ the process $\mathrm{N}_{\mathcal{N}_i^n}$ is responsive. Then the process $\mathrm{N}_{\mathcal{F},u}$ is also responsive.*

Proof. In any trace in which all events on input sockets are answered consider each event and its causality closure, then use the fact that the closure falls in a finite unfolding which is responsive to derive that the event is answered. □

Next, we establish that the property of being without hidden state propagates through recursive composition so that we can safely ignore differences between event and trace-based semantics.

Lemma 23 (No hidden state in recursively defined process). *If in a finitely unfolding family $\mathcal{F}$ all processes in all networks $\mathcal{N}_i$ are without hidden state, then the recursively defined process $\mathrm{N}_{\mathcal{F},u}$ is also without hidden state.*

Proof. Take any trace of the network $(\mathcal{F}, u)$ and find in it a causally-closed subtrace. Since direct causality in our processes and networks is finitely branching, the finitely unfolding property guarantees that there is a finite global bound for the unfolding to encompass the whole subtrace. Thus the subtrace is a valid trace of $\mathcal{N}_u^n$ for some n (Lemma 6) and therefore a valid trace of $(\mathcal{F}, u)$. □

7.4 Semantics of recursively defined networks and processes

Definition 24 (Semantics of recursively defined network). For a recursively defined network $(\mathcal{F}, u)$ without hidden state, we set:

$$[\![(\mathcal{F}, u)]\!]_{\mathrm{Q}} := \bigsqcup_{n\in\mathbb{N}} [\![\mathcal{N}_u^n]\!]_{\mathrm{Q}} \qquad [\![(\mathcal{F}, u)]\!]_{\mathrm{QA}} := \bigsqcap_{n\in\mathbb{N}} [\![\mathcal{N}_u^n]\!]_{\mathrm{QA}}$$

Lemma 25 (Semantics of recursively defined process). *For a finitely unfolding recursively defined network $(\mathcal{F}, u)$ without hidden state:*

$$[\![\mathrm{N}_{\mathcal{F},u}]\!]_{\mathrm{Q}} = \bigsqcup_{n\in\mathbb{N}} [\![\mathrm{N}_{\mathcal{N}_u^n}]\!]_{\mathrm{Q}} \qquad [\![\mathrm{N}_{\mathcal{F},u}]\!]_{\mathrm{QA}} = \bigsqcap_{n\in\mathbb{N}} [\![\mathrm{N}_{\mathcal{N}_u^n}]\!]_{\mathrm{QA}}$$

Proof. The lack of hidden state allows us to consider only traces of $(\mathcal{F}, u)$ that are causally-closed. Since the network is finitely unfolding and the direct causality in our networks is finitely branching, we have for each trace a number n such that the trace is valid also for $\mathcal{N}_u^n$. Thus in the trace-based definition of process semantics, we can map each causally-closed trace to a trace contributing to the expression of the right hand side, all being combined using the same operator, i. e. supremum for Q-semantics and infimum for QA-semantics. □

The following theorem extends Theorems 7 and 8.

Theorem 26 (Safety of recursively composed semantics).
For a recursive network family $\mathcal{F}$ without hidden state and $u \in I$:

$$[\![(\mathcal{F}, u)]\!]_{\mathrm{Q}} \upharpoonright_{S_{\mathcal{N}_u}} \sqsupseteq [\![\mathrm{N}_{\mathcal{F},u}]\!]_{\mathrm{Q}} \qquad [\![(\mathcal{F}, u)]\!]_{\mathrm{QA}} \upharpoonright_{S_{\mathcal{N}_u}} \sqsubseteq [\![\mathrm{N}_{\mathcal{F},u}]\!]_{\mathrm{QA}}$$

Proof. This theorem is a straightforward consequence of Lemma 25 and Def. 24 and the fact that both supremum and infimum are monotone operators. □

Theorem 27 (QA-consistency of recursively defined processes).
For a finitely unfolding responsive recursively defined network $(\mathcal{F}, u)$ in which all processes are QA-consistent, responsive and without hidden state:

$$[\![(\mathcal{F}, u)]\!]_{\mathrm{QA}} \upharpoonright_{S_{\mathcal{N}_u}} = [\![\mathrm{N}_{\mathcal{F},u}]\!]_{\mathrm{QA}}$$

and the process $\mathrm{N}_{\mathcal{F},u}$ is QA-consistent.

Proof. By Theorem 12, each encapsulated unfolding $\mathrm{N}_{\mathcal{N}_u^n}$ is QA-consistent and the QA-semantics equality holds. Therefore, the equality holds thanks to Lemma 25 and Def. 24. Each causally-closed trace reaches the QA-semantics because it can be contained within some unfolding $\mathrm{N}_{\mathcal{N}_u^n}$, proving that $\mathrm{N}_{\mathcal{F},u}$ is QA-consistent. □

8 Conclusion

We have laid the foundations of a framework in which one can study *distributed* query-answer based computation and is especially suitable for *exact* computation over real numbers and other similar continuous types — on ground or higher orders — as encountered in analysis and geometry. To achieve this goal, we have provided means to:

- analyse protocols for query-answer dialogues facilitating the transfer of partial information about objects;
- express the behaviour of a process that communicates with others using protocols of this kind, capturing it both at abstract trace and semantic levels;
- safely estimate the semantic behaviour of a finite or infinite process network from the behaviour of its components.

To carry out a *compositional* analysis of safety of finite nested networks, Theorems 7, 11 and 12 are provided. To apply Theorem 11, one has to demonstrate the Q-decreasing property — which requires an upper bound on the Q-semantics of a network — which is in turn provided compositionally according to Theorem 8. In summary, Theorems 7, 8, 11 and 12 together enable compositional reasoning about semantics and convergence rates of finite networks. These results extend well to nested networks that include recursively defined infinite compositions as we showed in Section 7.

8.1 Implementation overview

We have developed and published a BSD-licenced Haskell library [Konečný 2008a] for defining protocols, executable processes and finite as well as recursively defined infinite dataflow networks that fit the theory described in this article.

To enable protocols for exact real numbers and functions there are associated libraries for arbitrary precision interval arithmetic [Konečný 2008b] and outwards-rounded arbitrary precision multi-variate polynomial intervals [Konečný 2008c]. These libraries provide an abstract view of the arithmetic via type classes and give some choices for the back-ends as well as potential for extensions with new back-ends. For example, where fixed precision arithmetic is sufficient, one can opt for a back-end that uses `Double` values for interval endpoints or polynomial coefficients. One of the back-ends enables the use of the MPFR library arbitrary-precision floating point arithmetic.

Formally, a protocol is a pair of data structures — one for queries and one for answers — that implements the 2-parameter type class `QAProtocol`. All protocols that are provided are binary serialisable for efficient distributed communication. A process is defined as a data structure that contains information about input and output socket names and their protocols for static checking as well as a deployment function that starts up listener(s) for queries on output sockets and responds to them appropriately.

A network is defined as a data structure containing a list of named processes, channel identifiers with process and socket mappings. Protocol “type checking” is currently performed at network deployment time as follows. While channels are created with specific protocol types, they are stored in variables and passed in parameters that are typed with an existential type that permits any protocol. When a process deployment function receives its channels as parameters from a network deployer, it casts the channels parameters to the protocol types it expects. If the cast fails, the deployment function produces a detailed error message.

Nested and recursive networks are supported via a function that wraps a network as a process, in a manner similar to Definition 5. A process with no input sockets can be deployed and sent queries on its output sockets. Deployment is directed by a network manager. A network manager implements the type class `Manager` which has operations for

- constructing a new manager with a specific network name,
- connecting one manager with another manager,
- deploying a process on a given manager.

Process deployment may involve the deployment of subprocesses if the process is a wrapped network. The manager could (but currently does not) negotiate with other managers to distribute the subprocesses over several computers.

By the generality of the framework, it is possible to incorporate as individual processes certain existing programs, e. g. validated differential equation solvers such as [Rauh et al. 2007, Nedialkov 2006, Makino and Berz 2005]. Where two processes use different representations, they can be connected with the help of protocol converter processes.

8.2 Related work

Our model differs from the well studied nondeterministic dataflow network model [Jonsson and Kok 1989] mainly in the level of abstraction in its denotational semantics. In dataflow networks the discrete objects passing through channels are usually the data of interest and thus are not interpreted further. For us these objects are approximations of continuous data of interest. When we drop this interpretation and treat each channel as two one-directional channels, we get a special case of the usual nondeterministic dataflow networks. Thus various works on nondeterministic dataflow networks can be applied on our networks when viewed from such a low-level perspective.

Our work shares some aspects with some currently active projects. For example, project Erasmus [Grogono and Shearing 2008] defines a language for convenient programming of communicating processes. A subset of this language could directly express our kind of dataflow networks, including the definition of a concrete syntax for our protocols. While our current prototype is using Haskell, which is a mature high-level language, it may be interesting to develop another prototype in the latest version of MiniErasmus, which is a prototype of the full Erasmus language. Nevertheless, we are not aware of a denotational semantics for Erasmus that is similar to ours. As the Erasmus language supports also fully dynamic networks, an extension of our semantics to full Erasmus would be highly non-trivial.

In the *Ptolemy* project[3], some challenges have been tackled regarding the heterogeneity of the components of certain concurrent systems including dataflow networks similar to ours [Eker et al. 2003]. Compared with theirs, our perspective is more theoretical. In particular, there has been no (denotational) semantics developed for the Ptolemy framework similar to ours.

Most numerical programming occurs in languages that are designed for sequential execution, mainly Fortran, MATLAB and C++. Our approach allows one to package manageable components written in such languages and deploy them as processes in a network. In this way one can build systems and make use of existing *validated* numerical and geometrical tools and libraries, such as INTLAB. Our model can be a suitable abstraction above the low-level middleware such as MPI and Globus for programming concurrent and/or distributed scientific applications.

Notice that one can express a lazy version of the very successful *MapReduce* abstraction [Dean and Ghemawat 2008] in our model. Thus our paradigm can be seen as generalising map-reduce while keeping most of its semantic advantages and simplicity.

8.3 Future work

8.3.1 Further theorems for compositional analysis

A natural concept of *convergence rate* arises out of the current framework which demands further investigation along the lines of the results provided in this article regarding compositional analysis. Other important concepts requiring compositional analysis are those of communication and time complexity of processes and networks.

8.3.2 Many-valued computation

Based on the setting in this article, many-valued computation is given semantics that may hugely underestimate the actual results by only providing the infima of all possible answers. The recent advances in semantics of many-valued operations (e. g. [Marcial-Romero and Moshier 2008]) can provide a platform to develop more informative semantics for many-valued distributed computation. These include important problems such as root finding. No continuous single-valued map exists from the space of functions over real numbers to real numbers which returns the root of its input function. Moreover, once our framework is equipped with such semantics, it is

[3] http://ptolemy.eecs.berkeley.edu/

possible to make more use of other frameworks that include many-valued computation, such as Weihrauch's TTE [Weihrauch 2000], the languages of [Potts 1998] and [Marcial-Romero and Escardó 2007] and Müller's iRRAM [Müller 2001].

8.3.3 Concrete problems

We plan to design, verify, analyse and optimise networks for computational geometry operations, verification of hybrid systems and solving ordinary and partial differential equations. Some of these networks have been already implemented and show promise in tests but they still await formal analysis.

8.3.4 Distributed back-ends

While the library [Konečný 2008a] is designed with distributed deployment in mind, at present only deployment on one computer node is supported. We plan to provide back-ends that will enable an efficient deployment of exact dataflow networks on clusters as well as on specialised hardware with an MPI interface or on various grid environments.

Acknowledgements

We are very grateful for the insights and corrections raised by the anonymous reviewers. They helped us improve the presentation and focus of this article.

References

[Dean and Ghemawat 2008] Dean, J. and Ghemawat, S.: Mapreduce: Simplified data processing on large clusters; *Communications of the ACM*, 51(1):107–114, 2008.

[Eker et al. 2003] Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y.: Taming heterogeneity - the Ptolemy approach; *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 91(1):127–144, January 2003.

[Grogono and Shearing 2008] Grogono, P. and Shearing, B.: Concurrent software engineering: Preparing for paradigm shift; In *Canadian Conference on Computer Science and Software Engineering*, pages 99–108, May 2008.

[Jonsson and Kok 1989] Jonsson, B. and Kok, J. N.: Comparing two fully abstract dataflow models; In *PARLE '89: Proceedings of the Parallel Architectures and Languages Europe, Volume II: Parallel Languages*, pages 217–234, London, UK, 1989. Springer-Verlag.

[Konečný and Farjudian 2010] Konečný, M. and Farjudian, A.: Semantics of query-driven communication of exact values; *Journal of Universal Computer Science*, 16(18):2597–2628, 2010.

[Konečný 2008a] Konečný, M.: AERN-Net: Exact real networks; A Haskell library available at: `http://hackage.haskell.org/cgi-bin/hackage-scripts/package/AERN-Net`, November 2008.

[Konečný 2008b] Konečný, M.: AERN-Real: Arbitrary-precision interval arithmetic for approximating exact real numbers; A Haskell library available at: `http://hackage.haskell.org/cgi-bin/hackage-scripts/package/AERN-Real`, July 2008.

[Konečný 2008c] Konečný, M.: AERN-RnToRm: Arbitrary-precision arithmetic of multivariate piecewise polynomial enclosures; A Haskell library available at: `http://hackage.haskell.org/cgi-bin/hackage-scripts/package/AERN-RnToRm`, July 2008.

[Makino and Berz 2005] Makino, K. and Berz, M.: Cosy infinity version 9; *Nuclear Instruments and Methods A558*, pages 346–350, 2005.

[Marcial-Romero and Escardó 2007] Marcial-Romero, J. R. and Escardó, M. H.: Semantics of a sequential language for exact real-number computation; *Theor. Comput. Sci.*, 379(1-2):120–141, 2007.

[Marcial-Romero and Moshier 2008] Marcial-Romero, J. R. and Moshier, M. A.: Sequential real number computation and recursive relations; *Electron. Notes Theor. Comput. Sci.*, 202:171–189, 2008.

[Müller 2001] Müller, N. T.: The iRRAM: Exact arithmetic in C++; In *Selected Papers from the 4th International Workshop on Computability and Complexity in Analysis (CCA)*, volume 2064, pages 222–252. Springer-Verlag, 2001 Lecture Notes in Computer Science.

[Nedialkov 2006] Nedialkov, N. S.: Vnode-lp: A validated solver for initial value problems in ordinary differential equations; Technical Report CAS-06-06-NN, Department of Computing and Software, McMaster University, July 2006.

[Potts 1998] Potts, P. J.: *Exact Real Arithmetic Using Möbius Transformations* PhD thesis, University of London, Imperial College of Science, Technology and Medicine, Department of Computing, July 1998.

[Rauh et al. 2007] Rauh, A., Hofer, E. P., and Auer, E.: Valencia-ivp: A comparison with other initial value problem solvers; In *CD-Proc. of the 12th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics SCAN 2006, Duisburg, Germany, IEEE Computer Society*, page 36, 2007.

[Weihrauch 2000] Weihrauch, K.: *Computable Analysis, An Introduction* Springer-Verlag, 2000.