Journal of Universal Computer Science, vol. 16, no. 18 (2010), 2597-2628
submitted: 14/12/09, accepted: 18/7/10, appeared: 28/9/10 © *J.UCS*

Semantics of 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 address the question of how to communicate among distributed processes values such as real numbers, continuous functions and geometrical solids with arbitrary precision, yet efficiently. We extend the established concept of lazy communication using streams of approximants by introducing explicit queries. We formalise this approach using protocols of a *query-answer* nature. Such protocols enable processes to provide valid approximations with certain accuracy and focusing on certain locality as demanded by the receiving processes through queries.

A lattice-theoretic denotational semantics of channel and process behaviour is developed. The query space is modelled as a continuous lattice in which the top element denotes the query demanding *all the information*, whereas other elements denote queries demanding partial and/or local information. Answers are interpreted as elements of lattices constructed over suitable domains of approximations to the exact objects. An unanswered query is treated as an error and denoted using the top element.

The major novel characteristic of our semantic model is that it reflects the dependency of answers on queries. This enables the definition and analysis of an appropriate concept of *convergence rate*, by assigning an *effort indicator* to each query and a measure of information content to each answer. Thus we capture not only what function a process computes, but also how a process transforms the convergence rates from its inputs to its outputs. In future work these indicators can be used to capture further computational complexity measures.

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

Numerical computation is usually carried out in the floating-point model, which does not accurately represent the real numbers due to rounding errors. The discrepancy between real and floating-point results is studied mainly to find out how negligible the errors are. As an alternative, *exact real number computation* has been under active development for several decades (e. g. [Turing 1935, Gosper 1972, Weihrauch 1987] and more recently [Escardó 1997, Edalat and Potts 2000, Müller 2001, Lambov 2007]) providing results:

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

A. with explicit error bounds (*validated* computation),

B. to any arbitrarily high accuracy (*exact* computation).

Consequently, exact real programs are typically much easier to analyse and they can even be correct by construction. On the other hand, they tend to execute slower than their floating-point counterparts.

Some of the most practically successful approaches to validated computation are *interval arithmetic* [Moore 1966] and the so-called *Taylor models* [Revol et al. 2005, Neumaier 2003]. Note that computation using intervals with fixed-precision floating point numbers as end-points guarantees (A) but not (B).

1.1 Our goals

Practical approaches to exact real computation are mainly of two kinds: those emphasising compositional denotational semantics (e. g. [Escardó 1997, Edalat and Potts 2000]), and those emphasising execution speed (e. g. [Müller 2001, Lambov 2007]). One of our goals is to provide a framework for exact real computation that takes the best of the two approaches: *compositional denotational semantics* as well as the *practical speed-up* afforded by flexible representations and flexible programming models. For example, works from the former camp often support exact real computation via incrementality, i. e. the ability to extend previously given result by further digits, thus making it more and more accurate. Incrementality makes it particularly easy to compose exact computations as the computations can be seen as digit stream processors. On the other hand, in some cases incrementality leads to high resource usage due to the need to encode the previously given answers [Heckmann 1998]. In such cases, more efficiency could be obtained if computations are allowed to start from scratch when asked to give a more accurate approximation. Nevertheless, it is not as straightforward to compose computations that communicate their results non-incrementally.

Another important goal is to support practical exact computation where continuous and/or differentiable *real functions and geometrical objects*, such as solids and manifolds, are first-class data. This type of computation tends to be complex to program and computationally expensive to run, giving more weight to our first goal. Moreover, in certain instances this kind of computation can benefit from being *distributed*.

A computation that produces partial information about an exact result from partial information about exact parameters does so while producing and consuming the information at certain interdependent convergence rates. Our last goal is to provide a compositional analysis of *convergence rates* and make this notion available not only for real numbers but also for functional and geometrical data types.

1.2 Outline of our contribution

To achieve these goals, we propose an adaptation of dataflow process networks — pioneered by Kahn [Kahn 1974] — as they provide a model of distributed computation with simple denotational semantics that can be derived compositionally for nested systems. A Kahn dataflow network is a collection of processes connected by directed channels. The networks can be infinite if they are described recursively. During execution, the layout of channels is static and processes communicate via streams of symbols over buffered channels.

We build on the existing theory of Kahn dataflow networks by adding a more abstract semantics and a slightly more complex execution model. Our semantics adds appropriate data types to cater for computation with continuous data types such as real numbers, continuous real functions and geometrical solids. Kahn's semantics is at the level of symbol streams and therefore requires that the processes are deterministic at that level. Our processes are not deterministic but typically are extensional, i. e. deterministic at the level of our more abstract semantics. The execution model for our networks is stipulated to be a *lazy* one in which all communication is driven by explicit queries to facilitate a more efficient partial information flow.

Section 2 outlines these ideas together with examples. The rest of this article gives formal definitions of these concepts, focusing on individual channels and individual processes as outlined in Subsection 2.4.

1.3 Technical summary

The main technical contribution of this article is the derivation of domain-theoretic denotational semantics from query-answer trace semantics. The queries form the basis of a complete continuous lattice, each query representing a request for some partial information about an exact result. The top element represents an ultimate query for all information — it usually cannot be asked directly but via a sequence of smaller queries converging to it. The answers to queries are identified with a basis of a dcpo with an added top element that represents error. A trace is a set of query-answer pairs with an explicit causality relation. We deliberately ignore time interleaving. Initially we consider traces of communication between a value provider and value consumer and later we consider traces that include all communication of a process on its input and output sockets. In related work [Konečný and Farjudian 2010] we also consider traces of networks. The causality relation is used for processes to capture for each outgoing query the query or queries that caused it. Networks will introduce other sources of explicit causality.

Semantics for a simple two-party communication is derived from a set of traces capturing all possible interactions and it is a continuous function from the query lattice into the answer lattice. The difficulty is how to safely supply values for queries that do not appear in any trace. Our solution makes important use of the top element in the answer lattice. Semantics is then extended to processes that can be denotationally viewed as transformers of query-to-answer semantics of individual input/output connections.

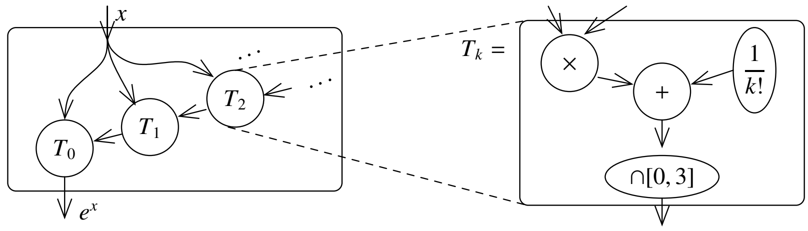

Figure 1: An infinite network computing the exact exponential on [0, 1], cf. [Potts 1998, p.124].

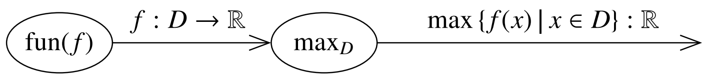

Figure 2: A simple network computing the maximum of a function f over an interval J.

2 Kahn's dataflow networks extended

2.1 Data types

Kahn's semantics is in terms of sequences of symbols, thus abstracting away the exact timing but *not* interpreting the sequences as representations of any higher-level data such as real numbers. Potts [Potts 1998] has used similar dataflow networks interpreting the sequences as exact real numbers, albeit restricted to a fixed representation and a very specific type of process. Our framework is more general than that of Potts' in the sense that it provides a formalism for dataflow networks with a wider variety of data types, processes and representations.

For example, we can have a simple real number network such as the one in Fig. 1. When executing, only a finite portion of the infinite network will be built and activated at each moment. Using a fixed-point method similar to that of Kahn, one can work out denotational semantics of finite portions of the network over the real interval domain (see e. g. [Escardó 1997]), replacing certain infinite parts with decoys that output no information, which is denoted as bottom in the semantic model. Then, using a domain theoretic limit, one extends the semantics to the whole infinite network. Using the Taylor theorem it can be proved that in such domain theoretic exact real number semantics the network outputs the exact exponential of the input real number x whenever $x \in [0, 1]$.

The network in Fig. 2 computes the maximum value of a fixed continuous function f over a compact domain D. *Information about the function is communicated over a channel to the process* $\max_D$ that contains a general maximum-finding algorithm for continuous functions of type $D \to \mathbb{R}$. In a yet unpublished paper, we have presented a rigorous proof demonstrating that, once considered over the space of all continuous functions of type $D \to \mathbb{R}$, an appropriate query-answer protocol provides superior overall performance for the maximisation network of Fig. 2, compared with a strict one-way protocol. With a query-answer protocol, at successive iterations, the maximising process $\max_D$ can ignore the areas guaranteed not to contain any maximum point, and instead focus the resources on the rest of the domain of search. This way, the maximum set of the function f — i. e. the set of all maximum points of f — is approximated together with the maximum value. Furthermore, we have quantified the improvement in the complexity of the communications once the query-answer protocol is employed, whereby the maximising node can specify which region of the function's domain it is interested in.

2.2 Convergence rates

The next step in adapting dataflow networks to exact computation is adding another kind of semantics. The semantics discussed so far tells us that the network in Fig. 1 outputs the exact exponential of the input. This type of denotational semantics completely hides the method of transferring the potentially infinite amount of information about particular real numbers or functions between the processes in the network. Assuming the transfer is gradual and one can measure the progress (e. g. using natural numbers), we may want to also capture the rates at which the transferred approximations converge[2]. In the case of the exponentiating network in Fig. 1, we may want to check that whenever the input number is provided with an exponential rate, the output number is also provided with an exponential rate. Assuming that all processes used in the network have this property, we claim that the whole network does. (Note that this is not true for all networks.)

2.3 Lazy communication

As our last step, we turn our attention to how the networks are executed. Most dataflow networks in the literature assume that communication over a channel is one-way, i. e. it is the supplier process that solely determines what symbols are sent over the channel. The programmer establishing a channel makes sure the data sent by the supplier process is the data the consumer process needs.

For objects such as functions and solids, the consumer may only need information on specific parts of the object, e. g. the graph of a function only on certain subdomains, with varying accuracies over each. The portion a consumer needs is (typically) very hard, or practically impossible, to be determined statically. We therefore let networks execute with two-way communication over channels so that consumers can make queries to indicate what they need. For simplicity, we require that each query receives at

[2] Note that the convergence rate is usually independent of computational complexity measures.

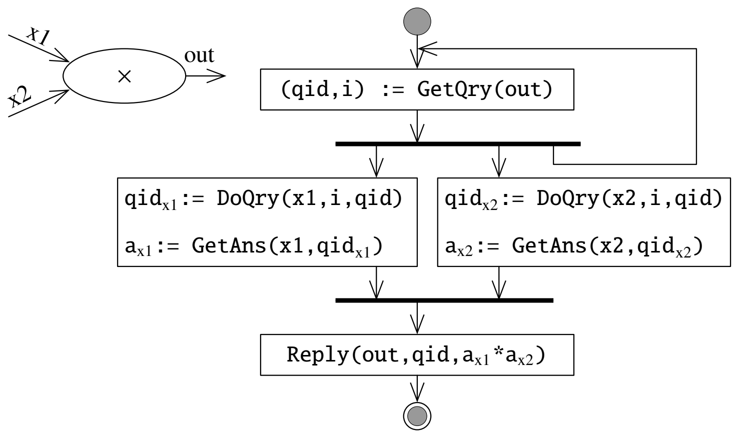

Figure 3: A process multiplying two real numbers communicated using intervals.

most one answer and nothing is sent from the supplier except an answer to a query. This execution model extends traditional laziness where the set of allowed queries is determined by the structure of the data, e. g. if the data is a pair, the query can only specify whether one needs the left or the right component. With functional or geometric data types, there is a much wider choice of meaningful queries and one has to restrict them explicitly as part of a *protocol*. For example, the maximising network in Fig. 2 may use a protocol, which we denote $\mathcal{P}_{\mathrm{Fn}}$, that allows e. g. the following interaction about the function $\lambda x.e^x$:

$$\begin{array}{lcl} ([0,1],2) & ?\!\mapsto\! ! & (0\overset{[0,5]}{\underline{\quad}}1) \\ ([0,2],2) & ?\!\mapsto\! ! & (0\overset{[0,9]}{\underline{\quad}}2) \\ ([1,2],3) & ?\!\mapsto\! ! & (1\overset{[2,9]}{\underline{\quad}}2) \\ ([0,2],3) & ?\!\mapsto\! ! & (0\overset{[0,3]}{\underline{\quad}}1\overset{[2,9]}{\underline{\quad}}2) \end{array} \tag{1}$$

where $?\!\mapsto\!!$ joins a query with its answer. Each query gives the provider a domain of interest (e. g. $[0,2]$ in the last pair) and a so-called *effort indicator*, i. e. a number indicating how hard to try to give a precise answer.[3] The answers are piecewise constant function enclosures, e. g.:

$$(f = 0\overset{[0,3]}{\underline{\quad}}1\overset{[2,9]}{\underline{\quad}}2) \quad \equiv \quad \big(\text{if } x \in [0,1] \text{ then } f(x) \in [0,3]; \text{if } x \in [1,2] \text{ then } f(x) \in [2,9]\big)$$

Fig. 3 shows how a process could be defined using a version of UML2 activity diagrams. Recall that the horizontal bars represent thread fork and join points. The backwards arrow from the fork starts a new thread ready to process the next request

[3] The value provider process may interpret the effort indicator as a desired accuracy. If the value converges, effort indicators can be always interpreted as accuracy indicators but may be very hard to determine what accuracy they indicate. We deal with this issue to some degree in Subsection 4.5.

concurrently. Note that our processes are also lazy: they cannot do anything observable until a query arrives because each outgoing query has to be justified by some incoming query. This is important when defining process semantics (see Subsection 5.3). The assignments in the code blocks use four primitive instructions: receive a query, send a query, receive an answer and send an answer. Query identifiers are used to link answers to queries and also to indicate which incoming query is the reason for an outgoing query.

Communication using queries and answers allows for a natural definition of convergence rate, as long as we define (cf. Subsection 4.5):

- what type of ideal result each query aims for;
- a measure of progress within a sequence of queries;
- a measure of quality for the answers relative to the aim of the queries.

2.3.1 Connection with game semantics

It is worth mentioning here that our query-answer approach is very similar to the way game semantics is constructed [Abramsky et al. 2000]. Our objective here is not to obtain mathematical structures for the semantics of our computation model through games, though the connection will definitely be explored in future work.

2.4 Outline of formalisation

In order to develop these ideas of ours into a framework suitable for design and implementation of practical exact computation, one needs to:

- define one or more languages in which one can precisely define various protocols for communicating approximations of exact values over a two-way channel;
- define one or more languages in which one can express precisely how a process receives and generates query and answer events for the channels that are connected to it through its sockets (e. g. the one used in Fig. 3);
- give functional semantics to the language(s) to express process behaviour in terms of:
  - transforming information from input to output channels as elements of high-level data types derived from real numbers,
  - transforming information as above but also capturing how processes react to and define channel communication convergence rates,
  - propagating queries from output to input channels to aid liveness analysis;

– provide a practical way to compute useful estimates for the above semantics for nested process networks in a compositional manner.

In this article we focus on the third step only. We omit the first two steps, i. e. we do not formally define any languages for defining protocols and processes. Instead we define channel protocols and process behaviour as sets of possible event traces. This will facilitate the inclusion of processes defined in a variety of languages as long as these languages have trace semantics compatible with our assumptions. Our traces are slightly more abstract than usual. We switch the usual linear ordering of events by time for a transitive causality relation, thus hiding the time interleaving of independent events, which would unnecessarily increase the number of traces if it had to be accounted for. The causality relation is used chiefly to capture when a query made to a process on one of its output channels explicitly results in the process making one or more queries on its input channels. One can have a query causing other queries around a loop in a network and by transitivity causing another query on the same channel. Thus we have to account for potential causality even when considering channels in isolation.

We first introduce our take on the concept of trace in Section 3. Then we consider protocols and the various semantics for traces of events on individual channels in Section 4 whilst in Section 5, we focus on semantics of individual processes.

3 Causality traces

A *trace* in our context is viewed as a set of *events* and their descriptions partially ordered by a *causality* relation. In practice, an event description usually consists of a query and either its corresponding answer or the symbol Ω when the query is unanswered.[4]

Definition 1 (Causality trace, Tr_Σ). A *causality trace* over a set Σ is a tuple $\zeta = (\Sigma, E, \eta, \rhd\!\!\bullet)$ where

– Σ is a set of *event descriptions*,

– E is a set of *events*,

– η: $E \to \Sigma$ assigns descriptions to events,

– $\rhd\!\!\bullet \subseteq E \times E$ is the *direct causality relation*, which is anti-reflexive and well-founded, i. e. there is no infinite chain $\cdots \rhd\!\!\bullet\, \beta_3 \rhd\!\!\bullet\, \beta_2 \rhd\!\!\bullet\, \beta_1$.
Whenever $\beta_1 \rhd\!\!\bullet\, \beta_2$, we say event β_1 *directly causes* event β_2.

We use $\rhd$ to denote the transitive closure of $\rhd\!\!\bullet$.
Let Tr_Σ be the set of all causality traces over Σ.

[4] It may seem a bit dangerous to amalgamate query and its answer as one event as they happen at different times. Nevertheless, as our traces deliberately do not capture time sequencing, separate timing is irrelevant. There seems to be no benefit in keeping queries and their answers separate.

Definition 2 (Subtrace, causally-closed subtrace). Any subset of events $E' \subseteq E$ in a causality trace $\zeta = (\varSigma, E, \eta, \mathrel{\rhd\!\!\bullet})$ determines a *subtrace*, denoted $\zeta{\restriction_{E'}}$, which is defined as the causality trace $(\varSigma, E', \eta{\restriction_{E'}}, \mathrel{\rhd\!\!\bullet}')$ where $\mathrel{\rhd\!\!\bullet}'$ is the binary relation on E' defined as

$$\beta_1' \mathrel{\rhd\!\!\bullet}' \beta_2' \Leftrightarrow \exists n \in \mathbb{N}, \beta_1, \ldots, \beta_n \in E \setminus E' : \beta_1' \mathrel{\rhd\!\!\bullet} \beta_1 \mathrel{\rhd\!\!\bullet} \beta_2 \mathrel{\rhd\!\!\bullet} \cdots \mathrel{\rhd\!\!\bullet} \beta_n \mathrel{\rhd\!\!\bullet} \beta_2'$$

(If $n = 0$ then the right hand side would simply be $\beta_1' \mathrel{\rhd\!\!\bullet} \beta_2'$.)

Whenever there are no $\beta' \in E', \beta \in E \setminus E'$ with $\beta' \mathrel{\rhd\!\!\bullet} \beta$ nor $\beta \mathrel{\rhd\!\!\bullet} \beta'$ then $\zeta{\restriction_{E'}}$ is called a *causally-closed subtrace* of ζ.

To cater for traces whose events are indexed by process socket names and process names, we need the following:

Notation 1 *For an indexed family of sets* $\{X_i \mid i \in I\}$*, its* discriminated union *and* Cartesian product *are denoted and defined as follows, respectively:*

$$\sum\nolimits_{i \in I} X_i \coloneqq \bigcup \Big\{ \{i\} \times X_i \;\Big|\; i \in I \Big\}, \quad \prod\nolimits_{i \in I} X_i = \Big\{ f : I \to \cup\{X_i \mid i \in I\} \;\Big|\; \forall i \in I : f(i) \in X_i \Big\}$$

Definition 3 (Indexed causality trace, restriction, projection). A causality trace ζ over the discriminated union $\varSigma = \sum_{i \in I} X_i$ is said to be *indexed* by I.

For each index subset $I' \subseteq I$ we define a *restriction* of $\zeta = (\varSigma, E, \eta, \mathrel{\rhd\!\!\bullet})$ to I', denoted

$$\zeta_{I'} = (\varSigma_{I'}, E_{I'}, \eta_{I'}, \mathrel{\rhd\!\!\bullet}_{I'})$$

as the subtrace $\zeta{\restriction_{E_{I'}}}$ further restricted to be over the set $\varSigma_{I'} = \sum_{i \in I'} X_i$ instead of $\varSigma$ and where $E_{I'}$ is the set of those events among E that are labelled with descriptions from $\varSigma_{I'}$. Thus, if we let $\gamma_{I'} : E_{I'} \hookrightarrow E$ and $\iota_{I'} : \varSigma_{I'} \hookrightarrow \varSigma$ to be inclusion maps of their respective domains into their codomains, then the following diagram commutes:

$$\begin{tikzcd}
E_{I'} \arrow[r, "\eta_{I'}"] \arrow[d, hook, "\gamma_{I'}"'] & \varSigma_{I'} \arrow[d, hook, "\iota_{I'}"] \\
E \arrow[r, "\eta"] & \varSigma
\end{tikzcd}$$

We call $\gamma_{I'}$ an *event inclusion map*.

The *projection* of ζ to the index $i \in I$ is the trace

$$\zeta_i = (X_i, E_i, \eta_i, \mathrel{\rhd\!\!\bullet}_i)$$

where $E_i = E_{\{i\}}$, $\mathrel{\rhd\!\!\bullet}_i = \mathrel{\rhd\!\!\bullet}_{\{i\}}$ and $\eta_i(\alpha)$ is the second component of the pair $\eta_{\{i\}}(\alpha) \in \{(i, c) \mid c \in X_i\}$. When the index set I is a singleton $\{i\}$, we simply write γ_i instead of $\gamma_{\{i\}}$.

An indexed causality trace ζ is sometimes referred to as an *interleaving* of all its projections $(\zeta_i)_{i \in I}$.

Fig. 4 illustrates the concept of trace interleaving. Note that the two dashed links between elements related by $\mathrel{\rhd\!\!\bullet}$ cannot be "inherited" by any of the projections, only its transitive consequence (the bold link) is inherited indirectly. Similarly, there can be causality pairs in ζ that are not inherited in any of the projections, even indirectly.

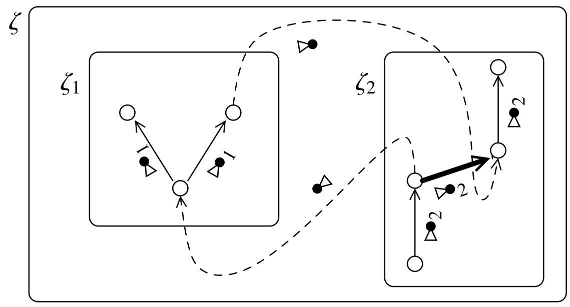

Figure 4: An interleaving ζ of two finite causality traces ζ_1 and ζ_2.

4 Query-answer protocols

As explained in Subsection 2.4, no concrete language for defining channel protocols is given here, nor any algebra of protocol operations. Instead we start from a low-level semantics in terms of trace sets and the way these sets are defined is left open. As a consequence, a constructive systematic mapping from protocols to semantic domains is impossible to develop and we will only require certain features and properties.

4.1 Protocol traces

We use causality traces to model channel protocol traces. Nevertheless, the causality relation plays no particular role at this stage. The relation will be used substantially when defining traces for processes and networks. For individual channels, the order is often inherited via projection from that of the processes and networks.

Definition 4 (Query-answer trace, $\mathrm{Tr}_{\mathfrak{Q},\mathfrak{A}}$). A *query-answer trace* over the set of queries $\mathfrak{Q}$ and the set of answers $\mathfrak{A}$ is a causality trace over the set $\Sigma_{\mathfrak{Q},\mathfrak{A}} := \mathfrak{Q}\times(\mathfrak{A}\cup\{\Omega\})$. We additionally require $\top \in \mathfrak{A}$, which denotes an explicit "error" and $\Omega \notin \mathfrak{A}$, which denotes a lack of answer.

We write $\mathrm{Tr}_{\mathfrak{Q},\mathfrak{A}}$ to stand for $\mathrm{Tr}_{\Sigma_{\mathfrak{Q},\mathfrak{A}}}$, i. e. all query-answer traces over the given sets.

Each event in a query-answer trace contains a query. When we focus on queries, we sometimes refer to events simply as queries. If such an event also contains an answer, we call it an *answered query*, otherwise an *unanswered query*.

In query-answer traces, causality $\beta_1 \rhd \beta_2$ where $\eta(\beta_1) = (q_1, x_1)$ and $\eta(\beta_2) = (q_2, x_2)$ is interpreted in two ways:

- occurrence β_1 of query q_1 causes occurrence β_2 of query q_2,
- occurrence β_2 of (non-)answer x_2 causes occurrence β_1 of (non-)answer x_1.

Definition 5. For a trace $\zeta = (\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathrm{Tr}_{\mathfrak{Q},\mathfrak{A}}$ we define:

- $\mathrm{Qry}(\zeta) := \{q \mid \exists x.(q, x) \in \eta(E)\}$ (the queries in ζ)
- $\mathrm{QA}(\zeta) := \eta(E) \cap (\mathfrak{Q} \times \mathfrak{A})$ (the query-answer pairs in ζ)
- $\mathrm{Ans}(\zeta) := \{a \mid \exists q.(q, a) \in \mathrm{QA}(\zeta)\}$ (the answers in ζ)

Definition 6. A *query-answer protocol* $\mathcal{P}$ is a triplet $(\mathfrak{Q}_\mathcal{P}, \mathfrak{A}_\mathcal{P}, \mathfrak{T}_\mathcal{P})$ where $\mathfrak{Q}_\mathcal{P}$ and $\mathfrak{A}_\mathcal{P}$ are sets, called its *query* and *answer sets*, respectively, and $\mathfrak{T}_\mathcal{P} \subseteq \mathrm{Tr}_{\mathfrak{Q}_\mathcal{P},\mathfrak{A}_\mathcal{P}}$ is a set of traces, called its *permissible traces*, satisfying the following property, which intuitively means that the protocol must allow both parties the freedom not to communicate:

For any $\zeta = (\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathfrak{T}_\mathcal{P}$ the following traces are in $\mathfrak{T}_\mathcal{P}$ as well:

- $(\Sigma, E, \eta, \rhd\!\!\bullet')$ for any anti-reflexive and well-founded $\rhd\!\!\bullet' \subseteq E \times E$
- $\zeta\restriction_{E'}$ for each $E' \subseteq E$ (subtrace)
- $(\Sigma, E, \eta', \rhd\!\!\bullet)$ where η' is obtained from η by turning some events to unanswered queries, i. e. for some $E' \subseteq E$ we have $\eta'(\beta) = \eta(\beta)$ for all $\beta \in E \setminus E'$, whilst for each $\beta' \in E'$, $\exists q \in \mathfrak{Q}_\mathcal{P}, a \in \mathfrak{A}_\mathcal{P} : \eta(\beta') = (q, a)$ and $\eta'(\beta') = (q, \Omega)$. (less answered trace)

Note that the empty trace is permissible in any protocol.

4.2 Protocol semantics

While data on channels is communicated as concrete symbols, we would like to interpret these symbols as representations of approximations that gradually communicate a more abstract semantic value. This concept is well-captured by domain-theory, which is the semantic model we use to interpret the abstract values being communicated.

Domain and lattice theory are rich theories, yet there are still varied — even opposing — terminology used even for very basic definitions. We adopt the terminology and definitions used by [Gierz et al. 2003].[5]

To minimise notation overhead, we will identify the query and answer symbols used in query-answer traces with their meaning as approximations to the 'perfect query' (i. e. "give me the whole object!") or approximation to a 'perfect answer' (e. g. a continuous real function). With this in mind, from now on we will require that for each query-answer protocol $\mathcal{P}$ we have:

- $\mathfrak{Q}_\mathcal{P}$ is a basis of a continuous complete lattice, which we denote $[\![\mathcal{P}]\!]_\mathrm{Q}$ and call the *query lattice*;

[5] In addition to [Gierz et al. 2003] the reader may refer to [Abramsky and Jung 1994, Amadio and Curien 1998] for extra information on the subject of domains and their use in semantics of programming languages.

– $\mathfrak{Q}_{\mathcal{P}}$ must not contain $\perp$, which is interpreted as "no query";

– $\mathfrak{A}_{\mathcal{P}}$ is a basis of a continuous complete lattice, which we denote $[\![\mathcal{P}]\!]_{\mathrm{A}}$ and call the *answer lattice*;

– $\top$ (i. e. the "error" answer introduced in Def. 4) is the top element of $[\![\mathcal{P}]\!]_{\mathrm{A}}$.

The maximal elements of the domain $[\![\mathcal{P}]\!]_{\mathrm{A}} \setminus \{\top\}$ are the ideal objects that the 'conversation' is about, i. e. the 'perfect answers'. For example, a protocol for continuous real functions similar to the one we considered informally on page 6 will have in its answer lattice a domain of continuous functions over the interval domain:

Definition 7 (Protocol $\mathcal{P}_{\mathrm{Fn}}$ for continuous real functions). The protocol $\mathcal{P}_{\mathrm{Fn}}$ has the following components:

– $\mathfrak{A}_{\mathcal{P}_{\mathrm{Fn}}} := \left\{ \bigsqcup_{i=1}^{n} (a_i \searrow b_i) \,\middle|\, n \in \mathbb{N},\ a_i, b_i \in \mathbb{IQ} \right\} \cup \{\top\} \subseteq [\mathbb{IR} \to \mathbb{IR}]^{\top} =: [\![\mathcal{P}_{\mathrm{Fn}}]\!]_{\mathrm{A}}$
where $(q \searrow a)(x) = a$ when $x \sqsupseteq_{\mathrm{I}} q$, otherwise $(q \searrow a)(x) = \perp$.

– $\mathfrak{Q}_{\mathcal{P}_{\mathrm{Fn}}} := \left\{ a_{\mathrm{L}} \overset{\mathrm{n}}{—} a_{\mathrm{R}} \,\middle|\, n \in \mathbb{N},\ a = [a_{\mathrm{L}}, a_{\mathrm{R}}] \in \mathbb{IQ} \right\} \subseteq [\mathbb{R} \to \omega_{\perp}^{\top}] =: [\![\mathcal{P}_{\mathrm{Fn}}]\!]_{\mathrm{Q}}$
where ω is the poset of naturally ordered natural numbers

– $\mathfrak{T}_{\mathcal{P}_{\mathrm{Fn}}} := \left\{ \zeta \,\middle|\, \zeta \in \mathfrak{T}_{\mathfrak{Q}_{\mathcal{P}_{\mathrm{Fn}}}, \mathfrak{A}_{\mathcal{P}_{\mathrm{Fn}}}},\ \bigsqcup \mathrm{Ans}(\zeta) \neq \top \right\}$

Notice that in this protocol the piece-wise constant function enclosures used as answers act on intervals, whereas on page 6 they acted on real numbers. Nevertheless, as the interval version is an extension of the real number version, it is justified to use the same notation for concrete queries and answers in this protocol as we did on page 6. In practise we may need more sophisticated variants of this protocol. For example, one can use piece-wise polynomial enclosures as implemented in [Konečný 2008] instead of piece-wise constant enclosures.

4.3 Extracting semantics from a set of traces

Next, we lift semantics of individual queries and answers to channel traces and sets of such traces so that we can describe the essence of channel communication for one or more variants of nondeterministic network execution. The higher the query, the more specific answer is required. Therefore, to safely approximate queries, we consider the least upper bound over alternative executions because we need an upper bound on the ideal type of answer the queries are asking for, e. g. when calculating convergence rates. To safely approximate answers, we use the greatest lower bound so that we obtain a lower bound on the information transferred in any execution variant.

Definition 8. The *Q-semantics of a trace* $\zeta \in \mathfrak{T}_{\mathcal{P}}$ is the element of $[\![\mathcal{P}]\!]_{\mathrm{Q}}$ defined as $[\![\zeta]\!]_{\mathrm{Q}} := \bigsqcup \mathrm{Qry}(\zeta)$. The Q-semantics of a set $Z \subseteq \mathfrak{T}_{\mathcal{P}}$ is $[\![Z]\!]_{\mathrm{Q}} := \bigsqcup_{\zeta \in Z} [\![\zeta]\!]_{\mathrm{Q}}$.

The query-to-answer mapping defined by a set of traces under protocol $\mathcal{P}$ can be expressed by a continuous function from $[\![\mathcal{P}]\!]_{\mathrm{Q}}$ to $[\![\mathcal{P}]\!]_{\mathrm{A}}$. From such a function one can find out whether under all circumstances the answers in the traces converge to the same object and if they converge, what object they converge to and a lower bound of the rate at which they converge. We denote the lattice of such functions $[\![\mathcal{P}]\!]_{\mathrm{QA}} := \left[[\![\mathcal{P}]\!]_{\mathrm{Q}} \to [\![\mathcal{P}]\!]_{\mathrm{A}}\right]$. To define the semantics of traces in this space, we first need the following auxiliary construction:

Definition 9. Let $(P_1, \sqsubseteq_1)$ and $(P_2, \sqsubseteq_2)$ be two posets and assume that P_2 has a top element $\top$. To each pair $(q, a) \in (P_1 \times P_2)$ a *co-step function* is assigned as defined by

$$(q \nearrow a) : P_1 \to P_2 \qquad (q \nearrow a)(x) := \begin{cases} a & \text{if } x \sqsubseteq_1 q \\ \top & \text{otherwise} \end{cases}$$

Notice that this is dual to the usual step function $(q \searrow a)(x)$ used in Def. 7.

Definition 10. The *QA-semantics* of a trace $\zeta \in \mathfrak{T}_{\mathcal{P}}$ is the element of $[\![\mathcal{P}]\!]_{\mathrm{QA}}$ defined as

$$[\![\zeta]\!]_{\mathrm{QA}} := \bigsqcap \left\{ q \nearrow a \;\middle|\; q = \textstyle\bigsqcup_{i \in I} q_i,\ a = \bigsqcup_{i \in I} a_i,\ \{(q_i, a_i) \mid i \in I\} \subseteq \mathrm{QA}(\zeta) \right\}$$

The QA-semantics of a set $Z \subseteq \mathfrak{T}_{\mathcal{P}}$ of traces is defined as $[\![Z]\!]_{\mathrm{QA}} := \bigsqcap \left\{ [\![\zeta]\!]_{\mathrm{QA}} \mid \zeta \in Z \right\}$. Trace A-semantics is defined as $[\![\zeta]\!]_{\mathrm{A}} := [\![\zeta]\!]_{\mathrm{QA}} ([\![\zeta]\!]_{\mathrm{Q}})$ and $[\![Z]\!]_{\mathrm{A}} := [\![Z]\!]_{\mathrm{QA}} ([\![Z]\!]_{\mathrm{Q}})$.

To help unpack these definitions, let us consider the semantics of the example finite trace from equation (1) on page 6. The meanings of its individual queries and answers are illustrated by diagrams in Fig. 5. Let us call this trace $\zeta_{\mathrm{exp,fin}}$. The Q-semantics $[\![\zeta_{\mathrm{exp,fin}}]\!]_{\mathrm{Q}}$ is the supremum of the four query elements, which is equal to the last one. The QA-semantics $[\![\zeta_{\mathrm{exp,fin}}]\!]_{\mathrm{QA}}$ maps the elements of the query lattice as shown in Fig. 6.

Our definition of QA-semantics does not seem natural at first sight. Next, let us justify the choices made and state a few key properties of this semantics. When computing QA-semantics of a set of alternative traces, we would like the result to be a safe lower bound for what was communicated in each trace:

Lemma 11 (QA-semantics safety). *For any set of traces Z and any query answer pair $(q, a) \in \zeta \in Z$, we have $[\![Z]\!]_{\mathrm{QA}}(q) \sqsubseteq a$.*

To get this property we need to use infimum when combining semantics of individual traces. When a specific query appears literally in multiple traces, it seems natural to assign it the infimum of all the answers given. This is the case for our definition except for the kind of undesirable behaviour when there is a higher query that is answered worse. When a single query appears twice in one trace, we could legitimately assign the better answer to the query in the semantics. Providing for such improvement would lead to a more complex definition. We avoid this as we view multiple occurrences of single queries in one trace as an undesirable behaviour. Thus we always take the infimum of

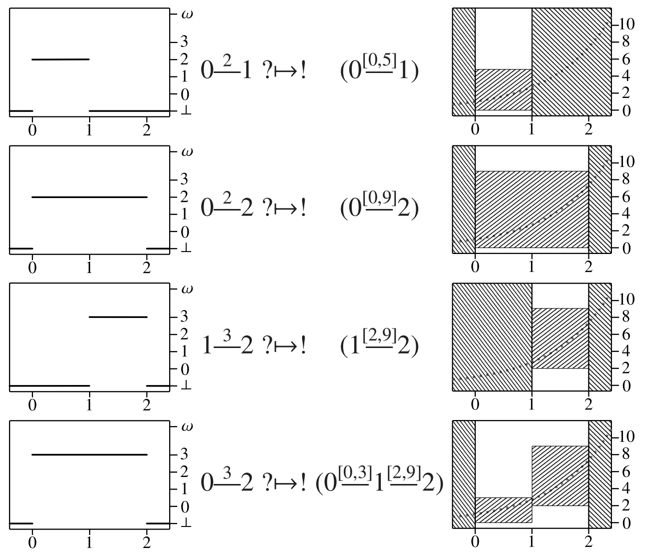

Figure 5: The semantics of individual queries and answers in the example trace $\zeta_{\text{exp,fin}}$, which communicates partial information about the function e^x using protocol $\mathcal{P}_{\text{Fn}}$.

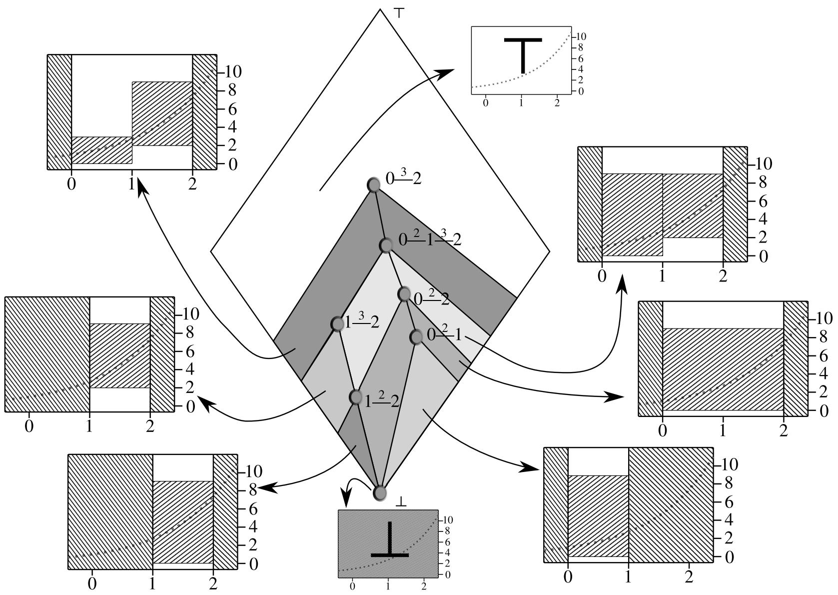

Figure 6: The QA-semantics of $\zeta_{\text{exp,fin}}$.

all answers whether in the same trace or in different traces as the semantics for a query that appears literally.

The main issue is what to assign to query elements that do not appear literally in the traces under consideration. It may seem natural that one should consider queries that are below the given element and use their answers as approximations. The problem is that it makes no sense to take the infimum of such approximations — it would always be near $\perp$. Thus we have no choice and take the opposite view and consider trace queries above the current query element and assign to this element the infimum of their answers. This is appropriate as long as we assume that the queries used in the formula are the only ones that could ever be asked on a certain channel. We will use trace QA-semantics only where this assumption is clearly true.

Let us now return to the example trace semantics. Notice that the bound $[0, 5]$ on domain interval $[0, 1]$ in the first query-answer pair does not play any role in the QA-semantics. This is so because its query is lower than the second query but its answer is higher than the second answer when restricted to $[0, 1]$. Whenever the first query-answer pair is relevant for computing QA-semantics for a particular query-lattice value, so is the second pair and due to taking an infimum of the co-step functions, the second answer will absorb the first answer over $[0, 1]$.

This behaviour seems to go against intuition as some extra information carried by the first pair is lost. If we want to avoid the loss, we need to either change the protocol or change the definition of QA-semantics. From the explanation following Lemma 11 it seems that one cannot define a safe QA-semantics otherwise without making it substantially more complicated. We concluded that the benefits obtained by adding an extra layer to the semantics do not justify the cost.

In our view, a better solution would be to use another interpretation of queries where the queries in our example are not comparable. The query lattice would be a variant of $\omega^{\mathbb{IQ}}$, i. e. the Cartesian product of a family — indexed by elements of $\mathbb{IQ}$ — of copies of ω. This leads to a different protocol, although syntax we used for the queries in $\mathcal{P}_{\mathrm{Fn}}$ can be re-used for specifying atomic queries in this new protocol and the answers would be exactly the same in both protocols. We do not develop this protocol here any further because it is harder to formalise and reason about than $\mathcal{P}_{\mathrm{Fn}}$ and its benefits over $\mathcal{P}_{\mathrm{Fn}}$ seem negligible.

4.4 Relative convergence

It seems natural to call a trace $\zeta \in \mathfrak{T}_{\mathcal{P}}$ *convergent* if $[\![\zeta]\!]_{\mathrm{A}}$ is a maximal element of the domain $[\![\mathcal{P}]\!]_{\mathrm{A}} \setminus \{\top\}$. This is all the more intuitive when we deal with ground types rather than function types, e. g. when $[\![\mathcal{P}]\!]_{\mathrm{A}}$ is $\mathbb{IR}$, the interval domain [Escardó 1997]. Nonetheless, in higher orders this definition is not adequate as there are cases where we are only interested in information locally. For instance, let $f : \mathbb{R} \to \mathbb{R}$ and assume that we are only interested in information about the graph of the function obtained by restricting the domain of f to $[0, 2]$. Thus, we consider a more general concept which

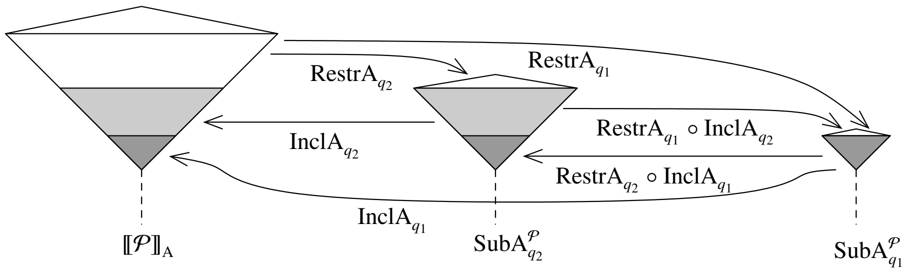

Figure 7: Illustration of adjunctions on semantic spaces relative to query elements $q_1 \sqsubseteq q_2$.

we call *relative convergence*. To that end, for each protocol $\mathcal{P}$ and each $q \in [\![\mathcal{P}]\!]_{\mathrm{Q}}$, we assume the existence of:

- a complete continuous lattice $\mathrm{SubA}^{\mathcal{P}}_q$ and
- a surjective adjunction $\mathrm{RestrA}^{\mathcal{P}}_q\colon\ [\![\mathcal{P}]\!]_{\mathrm{A}} \leftrightharpoons \mathrm{SubA}^{\mathcal{P}}_q\ :\mathrm{InclA}^{\mathcal{P}}_q$.

such that:

- for $q = \top$ the adjunction is bijective.
- for each $q_1 \sqsubseteq q_2$ in $[\![\mathcal{P}]\!]_{\mathrm{Q}}$, the following adjunction is surjective (see Fig. 7)

$$\mathrm{RestrA}^{\mathcal{P}}_{q_1} \circ \mathrm{InclA}^{\mathcal{P}}_{q_2}\colon\ \mathrm{SubA}^{\mathcal{P}}_{q_2} \leftrightharpoons \mathrm{SubA}^{\mathcal{P}}_{q_1}\ :\mathrm{RestrA}^{\mathcal{P}}_{q_2} \circ \mathrm{InclA}^{\mathcal{P}}_{q_1}$$

We shall omit the superscript $\mathcal{P}$ where protocol is clear from context.

The injective right adjunction InclA_q can be seen as *embedding* the lattice SubA_q as a sub-lattice *near the bottom* of the lattice $[\![\mathcal{P}]\!]_{\mathrm{A}}$ and RestrA_q can be seen as *projecting* every element in the larger lattice to a *lower* element in the sub-lattice. (Such adjunctions are called embedding-projection pairs.) Intuitively, for each query element q the adjunction determines a family of relevant answers, i. e. the maximal elements of $\mathrm{SubA}^{\mathcal{P}}_q \setminus \{\top\}$.

We now augment Def. 7 (i. e. protocol $\mathcal{P}_{\mathrm{Fn}}$) with these newly introduced protocol components so that we can talk about relative convergence of trace sets in this protocol:

- $\mathrm{SubA}^{\mathcal{P}_{\mathrm{Fn}}}_q := [\mathbb{I}S_q \to \mathbb{IR}]^\top$
  where $S_q = \{r \in \mathbb{R} \mid q(r) = \top\}$ and $\mathbb{I}S_q = \{a \mid a \in \mathbb{IR}, a \subseteq S_q\}$
- $\mathrm{InclA}^{\mathcal{P}_{\mathrm{Fn}}}_q(f) := \lambda a.(\text{if } a \in \mathbb{I}S_q \text{ then } f(a) \text{ else } \bot) : [\mathbb{IR} \to \mathbb{IR}],\quad \mathrm{InclA}^{\mathcal{P}_{\mathrm{Fn}}}_q(\top) := \top$
- $\mathrm{RestrA}^{\mathcal{P}_{\mathrm{Fn}}}_q(f) := \lambda a.f(a) : [\mathbb{I}S_q \to \mathbb{IR}],\quad \mathrm{RestrA}^{\mathcal{P}_{\mathrm{Fn}}}_q(\top) := \top$

For $\mathcal{P}_{\mathrm{Fn}}$ query-lattice elements that do not reach $\top$ for any real number, S_q is empty and the space $\mathrm{SubA}_q^{\mathcal{P}_{\mathrm{Fn}}}$ is the trivial two-element lattice, whose elements are mapped to $\bot$ and $\top$ by $\mathrm{InclA}_q^{\mathcal{P}_{\mathrm{Fn}}}$.

Definition 12 (Relative convergence). Let $\mathcal{P}$ be a protocol and $q_t \in [\![\mathcal{P}]\!]_{\mathrm{Q}}$. An element $d \in [\![\mathcal{P}]\!]_{\mathrm{QA}}$ is *convergent relative to* q_t iff $\mathrm{RestrA}_{q_t}(d(q_t))$ is *maximal* in $\mathrm{SubA}_{q_t}^{\mathcal{P}} \setminus \{\top\}$.

Moreover, d is *fully convergent* iff it is convergent relative to $\top \in [\![\mathcal{P}]\!]_{\mathrm{Q}}$.

A trace $\zeta \in \mathfrak{T}_{\mathcal{P}}$ is *relatively convergent* iff $[\![\zeta]\!]_{\mathrm{QA}}$ is convergent relative to $[\![\zeta]\!]_{\mathrm{Q}}$.

A set of traces Z is *relatively convergent* iff $[\![Z]\!]_{\mathrm{QA}}$ is convergent relative to $[\![Z]\!]_{\mathrm{Q}}$.

All finite $\mathcal{P}_{\mathrm{Fn}}$ traces have the trivial one-element target space and are therefore trivially relatively convergent. To get more interesting examples, consider the following $\mathcal{P}_{\mathrm{Fn}}$ traces:

- $\zeta_{\exp,[0,2]} = \left\{([0,2], n) \mathrel{?\!\!\mapsto\!!} \bigsqcup_{i=0}^{n}(0\frac{[1,1+2^{-i}]}{}2^{-i}) \;\middle|\; n \in \mathbb{N}\right\}$
- $\zeta_{\exp,0} = \left\{([0,2^{-n}], n) \mathrel{?\!\!\mapsto\!!} \bigsqcup_{i=0}^{n}(0\frac{[1,1+2^{-i}]}{}2^{-i}) \;\middle|\; n \in \mathbb{N}\right\}$

The function $[\![\zeta_{\exp,[0,2]}]\!]_{\mathrm{Q}}$ reaches $\top$ on the interval $[0,2]$ and thus aims to know the function fully on the whole interval $[0,2]$. On the other hand, $[\![\zeta_{\exp,0}]\!]_{\mathrm{Q}}$ reaches $\top$ only at point 0 and thus aims to know the function fully only at point 0. Both $[\![\zeta_{\exp,0}]\!]_{\mathrm{QA}}([\![\zeta_{\exp,0}]\!]_{\mathrm{Q}})$ and $[\![\zeta_{\exp,[0,2]}]\!]_{\mathrm{QA}}([\![\zeta_{\exp,[0,2]}]\!]_{\mathrm{Q}})$ give the same approximation, i. e. the enclosure that maps the singleton $[0,0]$ to $[1,1]$ and maps every other interval to a non-singleton interval or $\bot$. Thus $\zeta_{\exp,0}$ is relatively convergent whereas $\zeta_{\exp,[0,2]}$ is not. The set $\left\{\zeta_{\exp,[0,2]}, \zeta_{\exp,0}\right\}$ is not relatively convergent because the higher one of the two query-lattice elements is used to determine the target information and neither of the traces meets it. Please note that for a set to be relatively convergent, *all* the traces in it have to meet the common target. Also, a set of relatively convergent traces does not have to be convergent if the traces have different targets and do not reach the supremum of these targets.

Another interesting example is the trace that differs from $\zeta_{\exp,0}$ by replacing the supremum in the answers with only one term with $i = n$. The answers in the trace all agree that at point 0 the communicated function admits the value 1 but the answers also lose information close to point 0 while the query effort indicator increases. This leads in the limit to losing all information also about point 0 and the resulting semantics of the whole trace is the constant $\bot$ function. (The function that maps $[0,0]$ to $[1,1]$ and all other intervals to $\bot$ is not continuous and thus cannot be the infimum of the step functions that arise from this trace — the constant $\bot$ function is in fact the infimum.)

4.5 Convergence rates

Our next goal is to develop a notion of *convergence rate*. For that we need to quantify the information content in answers (i. e. the quality of answers) and extract from queries an indication of required answer quality. We will measure answer quality using real numbers extended with $+\infty$ for perfect answers (i. e. maximal elements of the answer domain) and indicate required quality using natural numbers extended with bottom and top. A convergence rate is a mapping from the latter to the former.

Thus, we require each protocol $\mathcal{P}$ to be equipped with:

- a *relative approximation quality measure* $|\cdot|_{\mathcal{P},q}\colon \mathrm{SubA}_q^{\mathcal{P}} \setminus \{\top\} \to \mathbb{R}^+_\infty$ satisfying
  - $a \sqsubseteq b \Rightarrow |a|_{\mathcal{P},q} \leq |b|_{\mathcal{P},q}$
  - $|a|_{\mathcal{P},q} = +\infty \Leftrightarrow a$ is a maximal element in $\mathrm{SubA}_q^{\mathcal{P}} \setminus \{\top\}$;
- a monotone query *effort indicator* map $|\cdot|_{\mathcal{P}}\colon [\![\mathcal{P}]\!]_{\mathrm{Q}} \to \omega^\top$ that takes $\bot$ to 0, $\top$ to $\top$ and no other element is mapped to 0.

We equip protocol $\mathcal{P}_{\mathrm{Fn}}$ with such measures as follows:

- $|f|_{\mathcal{P}_{\mathrm{Fn}},q} := |f|^{-1}_{\max}$ (i. e. the inverted maximum width of function enclosure f)
- $|q|_{\mathcal{P}_{\mathrm{Fn}}} = 1 + \max(q)$ (assuming that $1 + \bot = 0$ for convenience)

Definition 13 (Convergence rate). Let $\mathcal{P}$ be a protocol and $d \in [\![\mathcal{P}]\!]_{\mathrm{QA}}$ convergent relative to $q_{\mathrm{t}} \in [\![\mathcal{P}]\!]_{\mathrm{Q}}$.

The function

$$\mathrm{convRate}_{\mathcal{P},q_{\mathrm{t}}}(d) : \ \mathbb{N} \to \mathbb{R}^+_\infty \ : n \mapsto \left| \bigsqcap_{q \sqsubseteq q_{\mathrm{t}}, |q|_{\mathcal{P}} \geq n} \mathrm{RestrA}^{\mathcal{P}}_{q_{\mathrm{t}}}(d(q)) \right|_{\mathcal{P},q_{\mathrm{t}}}$$

is called the *convergence rate* of d measured relative to q_{t}.

For a relatively convergent trace ζ, $\mathrm{convRate}_{\mathcal{P}}(\zeta) := \mathrm{convRate}_{\mathcal{P},[\![\zeta]\!]_{\mathrm{Q}}}([\![\zeta]\!]_{\mathrm{QA}})$ and analogously for a relatively convergent set of traces Z, we define $\mathrm{convRate}_{\mathcal{P}}(Z) := \mathrm{convRate}_{\mathcal{P},[\![Z]\!]_{\mathrm{Q}}}([\![Z]\!]_{\mathrm{QA}})$.

We can now work out the convergence rate of the relatively convergent trace $\zeta_{\exp,0}$. In this trace, for a query whose second component is $n-1$, corresponding to an effort indicator n, we get an answer of size 2^{n-1} when focusing on the locus of convergence, i. e. point 0. Thus the rate of convergence for $\zeta_{\exp,0}$ is $\lambda n.2^{n-1}$. For a relatively convergent set of traces, its convergence rate is the infimum of the convergence rates of individual traces.

4.6 Protocols for higher order types

When communicating a value of a higher-order type such as a function operator, a query needs to specify a substantial amount of information such as a fairly accurate enclosure of a function or even an approximation of a higher-order function. While for order 2 it is still practical to send such an approximation as a part of the query, for orders 3 and more, it is usually not practical. We suggest to instead use a separate sub-channel going in the opposite direction for communicating the query information and have the query refer to this channel. Instead of formalising the creation of sub-channels and their references, we will simply pretend that a query can contain a trace of a sub-protocol. In practical implementations this trace would be realised on a sub-channel and could be shared among multiple queries. We consider this an optimisation that does not influence the semantics. When we consider computational complexity in future work, we may need to adapt the model to reflect sub-channels more closely.

Definition 14. A *higher-order query-answer protocol* is a query-answer protocol $\mathcal{P}$ with associated:

- *sub-protocol for queries* $\mathcal{P}^{\mathrm{S}}$
- *set of direct queries* $\mathfrak{Q}'_{\mathcal{P}}$
- a surjective *query-constructor* map $q^{\mathrm{HO}}_{\mathcal{P}} : \mathfrak{T}_{\mathcal{P}^{\mathrm{S}}} \times \mathfrak{Q}'_{\mathcal{P}} \to \mathfrak{Q}_{\mathcal{P}}$

Thus queries can be thought of as having two components: a direct component and a finite trace in the sub-protocol. When giving semantics to a higher-order query-answer protocol $\mathcal{P}$, it is recommended to first give semantics to the query sub-protocol $\mathcal{P}^{\mathrm{S}}$ and define the semantics of $\mathcal{P}$ queries with the help of the semantics of $\mathcal{P}^{\mathrm{S}}$ traces. One way to realise this idea is formalised in the following construction:

Definition 15 (Generic construction of higher-order protocol).
Assume query-answer protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ (which themselves may be higher-order).

The higher-order protocol $[\mathcal{P}_1 \to \mathcal{P}_2]$ has the following components:

- $\mathfrak{A}_{[\mathcal{P}_1 \to \mathcal{P}_2]} := \left\{ \bigsqcup_{i=1}^{n} (a_i \searrow b_i) \,\middle|\, n \in \mathbb{N}, a_i \in \mathfrak{A}_{\mathcal{P}_1}, b_i \in \mathfrak{A}_{\mathcal{P}_2} \right\} \cup \{\top\}$
  $\subseteq [[\![\mathcal{P}_1]\!]_{\mathrm{A}} \to [\![\mathcal{P}_2]\!]_{\mathrm{A}}] =: [\![[\mathcal{P}_1 \to \mathcal{P}_2]]\!]_{\mathrm{A}}$
- $[\mathcal{P}_1 \to \mathcal{P}_2]^{\mathrm{S}} := \mathcal{P}_1, \mathfrak{Q}'_{[\mathcal{P}_1 \to \mathcal{P}_2]} := \mathfrak{Q}_{\mathcal{P}_2}$
- $q^{\mathrm{HO}}_{[\mathcal{P}_1 \to \mathcal{P}_2]}(\zeta, q') := [\![\zeta]\!]^{\mathcal{P}_1}_{\mathrm{A}} \searrow q'$
  $\in \mathfrak{Q}_{[\mathcal{P}_1 \to \mathcal{P}_2]} \subseteq [[\![\mathcal{P}_1]\!]_{\mathrm{A}} \to [\![\mathcal{P}_2]\!]_{\mathrm{Q}}] =: [\![[\mathcal{P}_1 \to \mathcal{P}_2]]\!]_{\mathrm{Q}}$
- $\mathfrak{T}_{[\mathcal{P}_1 \to \mathcal{P}_2]} := \left\{ \zeta \,\middle|\, \zeta \in \mathfrak{T}_{\mathfrak{Q}_{[\mathcal{P}_1 \to \mathcal{P}_2]}, \mathfrak{A}_{[\mathcal{P}_1 \to \mathcal{P}_2]}}, \bigsqcup \mathrm{Ans}(\zeta) \neq \top \right\}$
- $\mathrm{SubA}^{[\mathcal{P}_1 \to \mathcal{P}_2]}_q := \left\{ F \in [[\![\mathcal{P}_1]\!]_{\mathrm{A}} \to [\![\mathcal{P}_2]\!]_{\mathrm{A}}] \,\middle|\, \forall a \in [\![\mathcal{P}_1]\!]_{\mathrm{A}} . F(a) \in \mathrm{SubA}^{\mathcal{P}_2}_{q(a)} \right\} \cup \{\top\}$
- $\mathrm{InclA}^{[\mathcal{P}_1 \to \mathcal{P}_2]}_q(F) := \lambda a. \left(\mathrm{InclA}^{\mathcal{P}_2}_{q(a)}(F(a)) \right) \qquad \mathrm{InclA}^{[\mathcal{P}_1 \to \mathcal{P}_2]}_q(\top) := \top$
- $\mathrm{RestrA}^{[\mathcal{P}_1 \to \mathcal{P}_2]}_q(F) := \lambda a. \left(\mathrm{RestrA}^{\mathcal{P}_2}_{q(a)}(F(a)) \right) \qquad \mathrm{RestrA}^{[\mathcal{P}_1 \to \mathcal{P}_2]}_q(\top) := \top$
- $|F|_{[\mathcal{P}_1 \to \mathcal{P}_2], q} := \min_{a \in [\![\mathcal{P}_1]\!]_{\mathrm{A}}} \left(|F(a)|_{\mathcal{P}_2, q(a)} \right)$

– $|q|_{[\mathcal{P}_1 \to \mathcal{P}_2]} \coloneqq \max_{a \in [\![\mathcal{P}_1]\!]_{\mathrm{A}}} \left(|q(a)|_{\mathcal{P}_2} \right)$

We shall now briefly study an example higher-order query-answer protocol for communicating continuous real function operators, which is defined using an instance of the above construction:

Definition 16 (Protocol $\mathcal{P}_{\mathrm{FnFn}}$ for continuous real function operators).
$\mathcal{P}_{\mathrm{FnFn}} \coloneqq [\mathcal{P}_{\mathrm{Fn}} \to \mathcal{P}_{\mathrm{Fn}}]$.

As an example, consider the following $\mathcal{P}_{\mathrm{FnFn}}$ trace, called $\zeta_{(\mathrm{f}-1)(1)}$, which communicates partial information about the function operator $\lambda f.\lambda x.f(x-1)$:

$$\left\{ q^{\mathrm{HO}}_{\mathcal{P}_{\mathrm{FnFn}}}(\zeta_n, ([1, 1+2^{-n}], n)) \ ?{\mapsto}! \ (A_n(0) \searrow A_n(1)) \,\middle|\, n \in \mathbb{N} \right\}$$

where $\zeta_n = \left\{ \left(([0, 2^{-i}], i) \ ?{\mapsto}! \ A_n(0) \right) \,\middle|\, 1 \leq i \leq n \right\}$ are finite prefixes of a unique $\mathcal{P}_{\mathrm{Fn}}$ trace, which we call ζ, that communicates that the function in question has value close to 1 near point 0 and $A_n(t) = \bigsqcup_{i=1}^{n} (t \xrightarrow{[1,1+2^{1-i}]} (t + 2^{-i}))$ is an enclosure that specifies that a function's value is close to 1 in the neighbourhood of t. Notice that $[\![\zeta_n]\!]_{\mathrm{A}} = A_n(0)$. Thus our trace $\zeta_{(\mathrm{f}-1)(1)}$ communicates the fact that the operator takes a function whose value is close to 1 at point 0 to a function whose value is close to 1 at point 1.

The semantics $q_{\mathrm{D}} \coloneqq [\![\zeta_{(\mathrm{f}-1)(1)}]\!]_{\mathrm{Q}}$ is a map that takes each function enclosure:

– that is not above $A_1(0)$ to $\perp$

– that is above $A_n(0)$ but not above $A_{n+1}(0)$ to $\bigsqcup_{i=1}^{n}([1, 1+2^{-i}] \searrow i)$

– that is above $A_\infty(0)$ to $\bigsqcup_{i=1}^{\infty}([1, 1+2^{-i}] \searrow i)$

Any enclosure a that falls into the first two cases above has trivial $\mathrm{SubA}^{\mathcal{P}_{\mathrm{Fn}}}_{q_{\mathrm{D}}(a)}$ and therefore only enclosures that fall into the third case can be given non-trivial constraints within $\mathrm{SubA}^{\mathcal{P}_{\mathrm{FnFn}}}_{q_{\mathrm{D}}}$. Moreover, the constraints are limited by the fact that for such lucky enclosures a, $\mathrm{SubA}^{\mathcal{P}_{\mathrm{Fn}}}_{q_{\mathrm{D}}(a)}$ is only registering the value of the resulting enclosure at point 1. This means that our trace $\zeta_{(\mathrm{f}-1)(1)}$ is relatively convergent because its A-semantics map function enclosures in the third category to the enclosure $A_\infty(1)$, whose restriction to $\mathrm{SubA}^{\mathcal{P}_{\mathrm{Fn}}}_{q_{\mathrm{D}}(a)}$ is a maximal element in this domain because its value at point 1 is exact. The convergence rate of the trace is $\lambda n.2^{n-1}$.

As discussed earlier, in a practical implementation it is desirable to optimise communication in higher order protocols, e. g. the trace components of the queries would not be sent with the query directly but communicated over a private channel. In the case of trace $\zeta_{(\mathrm{f}-1)(1)}$ the channel should be shared for all queries as they are prefixes of one another. Also the answers could be substantially compressed owing to the fact that part of the answer can be reconstructed from the query.

5 Processes

Processes communicate with each other through *channels* connected to the process' *sockets*. As explained in Subsection 2.4, we do not develop a particular language or calculus for defining processes. Instead, we capture their behaviour by observing the way they participate in the generation of queries and answers through their sockets. A functional denotational semantics for processes will be inferred based on the way events appear on their input and output sockets.

5.1 Process traces

Informally, a process trace is an interleaving of protocol traces indexed by the names of sockets, which are divided into *input* and *output* sockets. The intuition behind points (iii) and (iv) in Def. 17 below is that we make a distinction between *internal* and *external* events on a process trace. A process has complete control over its internal events but no control over the external ones, which can only be *discovered*.

Definition 17. A *process* (sometimes called a 'node') N comprises:

- a pair of disjoint sets of *socket names*: S^-_{N} (input) and S^+_{N} (output), with notation for the set of all sockets: $S_{\mathrm{N}} \coloneqq S^-_{\mathrm{N}} \cup S^+_{\mathrm{N}}$,
- a *protocol type* assignment, i. e. an assignment of a query-answer protocol $\mathcal{P}^s_{\mathrm{N}}$ to each socket $s \in S_{\mathrm{N}}$,
- a set of valid *process traces* $\mathfrak{T}_N$ such that:
  (i) Each trace $\zeta = (\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathfrak{T}_{\mathrm{N}}$ is an interleaving of its projections $(\zeta_s)_{s \in S_{\mathrm{N}}}$ (see Def. 3) where
    - each ζ_s is a valid trace in the protocol $\mathcal{P}^s_{\mathrm{N}}$;
    - each query on an input socket is directly caused by at least one query on an output socket;
    - there is no *direct* causality among events on input sockets, nor on output sockets, i. e. $\rhd\!\!\bullet\ \restriction_{E_{S^-_{\mathrm{N}}} \times E_{S^-_{\mathrm{N}}}} = \rhd\!\!\bullet\ \restriction_{E_{S^+_{\mathrm{N}}} \times E_{S^+_{\mathrm{N}}}} = \emptyset$.

  (ii) A process cannot influence the way its queries on input sockets cause some queries on its output sockets (e. g. via a network loop), i. e. whenever
    - $\zeta = (\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathfrak{T}_{\mathrm{N}}$ and
    - $\rhd\!\!\bullet \setminus (E_{S^-_{\mathrm{N}}} \times E_{S^+_{\mathrm{N}}}) = \rhd\!\!\bullet' \setminus (E_{S^-_{\mathrm{N}}} \times E_{S^+_{\mathrm{N}}})$

    then also $(\Sigma, E, \eta, \rhd\!\!\bullet') \in \mathfrak{T}_{\mathrm{N}}$.

(iii) A process cannot prevent its neighbours connected via output sockets to remain silent, nor can it control the way they make queries.

(iv) Similar to (iii), a process cannot prevent any partners connected via input sockets to remain silent, nor can it control the way they answer queries.[6]

A process can be given in a concrete language, such as that of Fig. 3, if its extracted trace set meets (i)–(iv).

As an example, consider the process defined in Fig. 3. It is supposed to multiply two real numbers communicated through its two input sockets named ‘x1’ and ‘x2’ and communicate the result through its only output socket named ‘out’. Assume all of the process’ sockets use the protocol for exact real numbers in which each query is a natural number with semantics in $\omega_\perp^\top$ and each answer is an interval with rational endpoints with semantics in $\mathbb{IR}$. The following is an example finite trace in the process’ trace set:

$$\begin{array}{ccc}
(\mathrm{out}, (\underline{1}, [0, 2])) & (\mathrm{out}, (\underline{1}, [0, 2])) & (\mathrm{out}, (\underline{2}, [1/2, 1\,1/2])) \\
(\mathrm{x1}, (1, \underline{\underline{[0, 1]}})) & (\mathrm{x1}, (1, \underline{\underline{[0, 1]}})) & (\mathrm{x1}, (2, \underline{\underline{[1/2, 1]}})) \\
(\mathrm{x2}, (1, \underline{\underline{[1, 2]}})) & (\mathrm{x2}, (1, \underline{\underline{[1, 2]}})) & (\mathrm{x2}, (2, \underline{\underline{[1, 1\,1/2]}}))
\end{array}$$

We see three queries on the output socket, each causing a pair of similar events on the input sockets. Moreover, one of the events on the input sockets caused via means unknown to the process two of the three incoming queries. By rule (ii) in the above definition, if we remove these two causalities from the trace, we get another valid trace. Also if we add any other such causality from output queries to input queries, the trace will be valid for the process as long as the causalities do not form a cycle as this would break the definition of causality trace (Def. 1).

Also, notice that the left and middle queries have the same content, being two occurrences of the same query element. If the process would cache its answers the trace could show the middle query without any consequences, i. e. the two events underneath it would be removed, assuming the middle query occurred after the left query had been answered. Alternatively, the middle and left ‘out’ queries can both individually cause the same pair of ‘x1’ and ‘x2’ queries.

The underlined bits are the ones given externally. Thus by rules (iii-iv), there have to be traces that take into account any other set of queries (potentially with very different consequent events) and for the same queries, there have to be traces that consider

[6] We omit the full formalisation of these two rules. The formalisation is technically complex but intuitively straightforward: Each event is broken down into bits that are determined by the current process and bits that are determined by its environment. The process must not make any assumptions about the bits that are given by the environment except that all protocols are followed. This is expressed by stating that for each trace in the trace set many other traces also have to be in the set — namely those traces that differ from this trace in the externally given bits. Moreover, the process has the freedom to adapt its own bits that depend on the changed external bits.

different answers to the queries on input sockets, potentially giving different answers to the queries on output sockets, or even generating further queries on input sockets before answering. Furthermore, there have to be other traces that capture what happens when some or all of the queries on the input sockets are not answered.

5.2 Further example processes

To get further insight, we introduce further two processes that in our experience tend to appear often in networks. Their activity diagram definitions are in Fig. 8. $N^{\mathrm{merge}_{\mathcal{P}}}$ has two input sockets and one output socket of the same protocol $\mathcal{P}$ and it merges information from both input sockets to the output socket, assuming the two input sockets provide information consistent with each another. $N^{\mathrm{cache}_{\mathcal{P}}}$ behaves as a cache of previously sent information to minimise repetition of queries. It can be inserted into any channel without changing the network denotational semantics but improving its operational behaviour. We usually omit its occurrences in network diagrams for simplicity.

Note that both of these processes are defined generically for any protocol $\mathcal{P}$. This is possible thanks to the generic availability of concepts such as effort indicator for queries and lattice-theoretical semantics for answers. Nevertheless, if one wants to instantiate such processes to a concrete protocol in a computable manner, one has to provide an effective representation of these concepts.

5.3 Process semantics

5.3.1 Process Q-semantics

To make the study of processes tractable, they are usually viewed functionally, i. e. as transformers, generating output based on what they receive as input. When focusing on queries instead of answers, processes act as transformers in the opposite direction. The following definitions formalise these two functional views of processes in the universe of our query and answer lattices.

We present two slightly different definitions of semantics. One is based on traces and one on individual events within traces. In many cases these are equivalent but not always. When focusing on events, we may capture situations that occur only in a certain context, such as when the process has accumulated certain knowledge based on previous requests, separating only certain events from that context even though they may be influenced by the context. On the other hand, when using traces, we cannot separate groups of events unless the groups can be split into two valid traces, i. e. when the groups are *de facto* independent. We will give examples of the differences after the definitions are established.

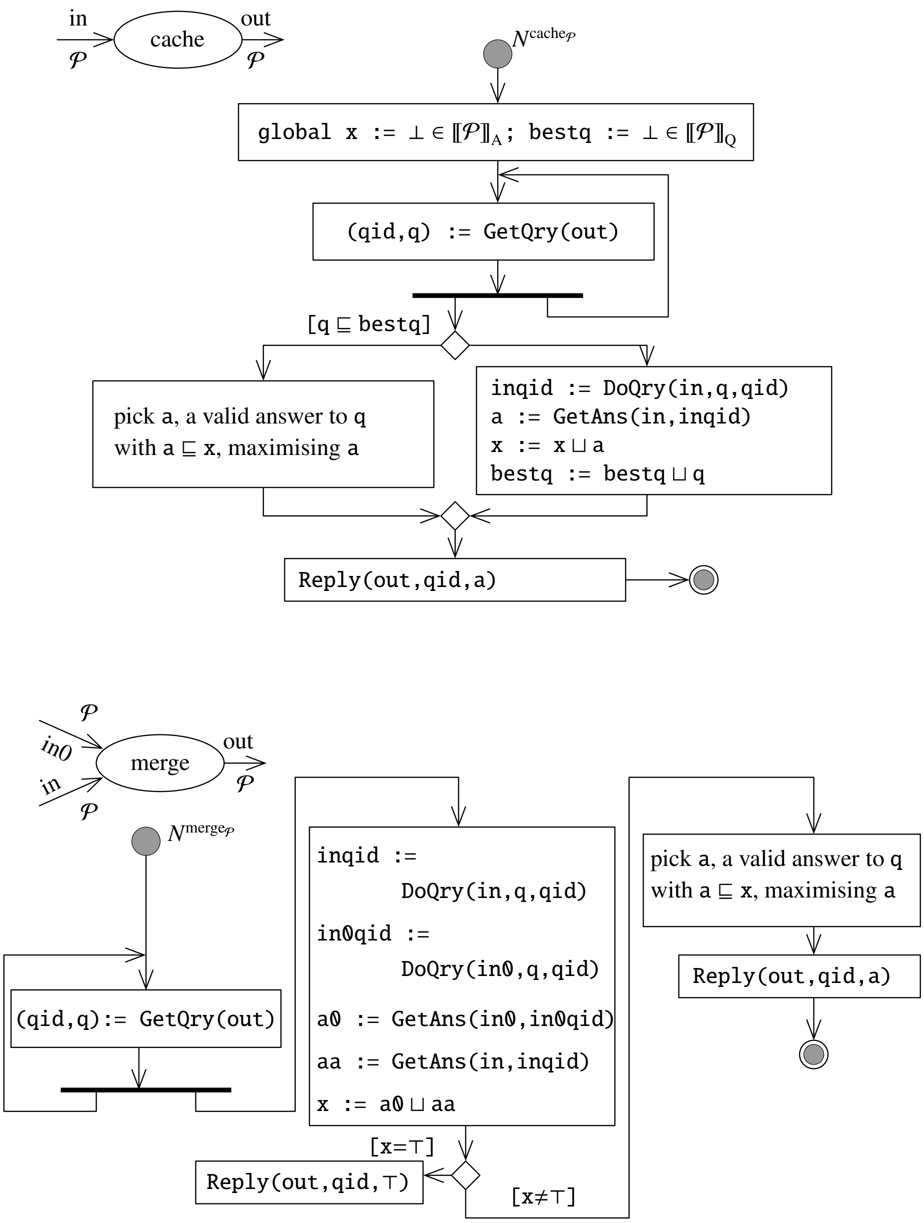

Figure 8: Definitions of processes $N^{\text{cache}_\mathcal{P}}$ and $N^{\text{merge}_\mathcal{P}}$.

Definition 18. The *event-based Q-semantics* of a process N for an output socket s^+ is defined as:

$$[\![\mathrm{N}]\!]_{\mathrm{Q}}^{s^+} : \left[\!\left[\mathcal{P}_{\mathrm{N}}^{s^+}\right]\!\right]_{\mathrm{Q}} \to \prod_{s^- \in S_{\mathrm{N}}^-} \left[\!\left[\mathcal{P}_{\mathrm{N}}^{s^-}\right]\!\right]_{\mathrm{Q}}$$

$$\left([\![\mathrm{N}]\!]_{\mathrm{Q}}^{s^+}(q_{s^+})\right)_{s^-} :=$$

$$\bigsqcup\Big\{q^- \Big| \exists(\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathfrak{T}_{\mathrm{N}}, \beta^+, \beta^-, q^+, x^+, x^- :$$

$$\eta(\beta^-) = (s^-, (q^-, x^-)), \eta(\beta^+) = (s^+, (q^+, x^+)),$$

$$\beta^+ \rhd\!\!\bullet\, \beta^-, q^+ \sqsubseteq q_{s^+}\Big\}$$

i. e. the function that given an upper bound on what has been asked on the *output* socket s^+, for each of its input sockets s^-, returns the least upper bound on what the process consequently asks through this input socket.

The event-based Q-semantics of N on *all output sockets* is defined as:

$$[\![\mathrm{N}]\!]_{\mathrm{Q}}^{\mathbf{E}} : \prod_{s^+ \in S_{\mathrm{N}}^+} \left[\!\left[\mathcal{P}_{\mathrm{N}}^{s^+}\right]\!\right]_{\mathrm{Q}} \to \prod_{s^- \in S_{\mathrm{N}}^-} \left[\!\left[\mathcal{P}_{\mathrm{N}}^{s^-}\right]\!\right]_{\mathrm{Q}}$$

$$[\![\mathrm{N}]\!]_{\mathrm{Q}}^{\mathbf{E}}\left((q_{s^+})_{s^+ \in S_{\mathrm{N}}^+}\right) := \bigsqcup_{s^+ \in S_{\mathrm{N}}^+} [\![\mathrm{N}]\!]_{\mathrm{Q}}^{s^+}(q_{s^+})$$

The trace-based definition is much simpler:

Definition 19. The *trace-based Q-semantics* of a process N is defined as:

$$[\![\mathrm{N}]\!]_{\mathrm{Q}}^{\mathbf{T}}\left((q_{s^+})_{s^+ \in S_{\mathrm{N}}^+}\right) := \bigsqcup\left\{\left([\![\zeta_{s^-}]\!]_{\mathrm{Q}}\right)_{s^- \in S_{\mathrm{N}}^-} \;\middle|\; \zeta \in \mathfrak{T}_{\mathrm{N}}, \forall s^+ \in S_{\mathrm{N}}^+ : [\![\zeta_{s^+}]\!]_{\mathrm{Q}} \sqsubseteq q_{s^+}\right\}$$

Lemma 20. *For any process* N*, we have* $[\![\mathrm{N}]\!]_{\mathrm{Q}}^{\mathbf{T}} \sqsubseteq [\![\mathrm{N}]\!]_{\mathrm{Q}}^{\mathbf{E}}$.

Proof. Starting with Def. 19, we note that for each trace ζ considered in the set construction, we have only queries below q_{s^+} on each output socket s^+ and all queries on input sockets are caused by some queries on output sockets, thus $\left([\![\zeta_{s^-}]\!]_{\mathrm{Q}}\right)_{s^- \in S_{\mathrm{N}}^-}$ is accounted for in $[\![\mathrm{N}]\!]_{\mathrm{Q}}^{\mathbf{E}}$ via the queries in ζ. □

As noted earlier, the two Q-semantics may differ. A simple example where it is the case is a process that passes a continuous real function using the $\mathcal{P}_{\mathrm{Fn}}$ protocol from its one input socket into its one output socket enforcing the 2^n convergence rate. Moreover, assume that the process is optimised to record past queries and when asked for a higher accuracy tries to guess the query in order to avoid having to make many queries while waiting for the accuracy of the input to reach the required level. Thus we may see the following queries: $([0, 1], 1)$ on output causing copies of the same query but with effort indicators $1, 2, 3, 5, 10$ on the input, then $([1, 2], 2)$ on the output, causing query

$([1,2],20)$ on the input. If the same process receives query $([1,2],2)$ without any history, it results in sending multiple queries with effort indicators $1,2,3,5,10,17$ to the input socket. This means that trace-based Q-semantics maps $([1,2],2)$ to $([1,2],17)$, whereas the event-based Q-semantics maps it to $(([1,2]),20)$ or higher because while studying what happens when $([1,2],2)$ arrives, it considers the first trace and its last two events ignoring the first six events and thus ignoring their learning effect.

It is desirable to identify a natural class of processes where the two semantics coincide. From our example it appears that one has to forbid learning of a certain nature. We capture such a property as being able to replay any causally-closed sub-trace of a valid trace, i. e. responding the same with or without context that is not connected via the causality relation:

Definition 21. A process N is said to be *without hidden state* iff for each trace $\zeta \in \mathfrak{T}_{\mathrm{N}}$ any causally-closed subtrace $\zeta\restriction_{E'}$ is also in $\mathfrak{T}_{\mathrm{N}}$.

In our example we can expose the process' hidden state by modifying it to be more honest about which outgoing query supports which incoming query — if all the past queries inform future answers, then they should all be seen as "caused by" the future queries.

Lemma 22. *For any process* N *without hidden state, we have* $[\![\mathrm{N}]\!]^{\mathbf{T}}_{\mathrm{Q}} = [\![\mathrm{N}]\!]^{\mathbf{E}}_{\mathrm{Q}}$.

Proof. Starting with Def. 18, for each trace with query $q^+ \sqsubseteq q_{s^+}$ we take the causally-closed subtrace generated by this event. Moreover, by point (ii) in Def. 17 we can assume that this subtrace contains no other query on an output socket. This subtrace is a valid process trace because the process is without hidden state. Such subtraces will account for all pairs q^+, q^-, so we get $[\![\mathrm{N}]\!]^{\mathbf{T}}_{\mathrm{Q}} \sqsupseteq [\![\mathrm{N}]\!]^{\mathbf{E}}_{\mathrm{Q}}$, which together with Lemma 20 gives the result. $\square$

We use the notation $[\![\mathrm{N}]\!]_{\mathrm{Q}}$ whenever we know that N is without hidden state.

From now on we will assume that all unspecified processes are without hidden state. All processes we considered so far are such except N^{cache}. We change N^{cache} to expose its hidden state by declaring that each of its queries is directly caused not only by the first query that triggered it but also by all subsequent queries that are informed by it via the cache. Note that in an infinite trace one query going out on an input socket may thus be formally caused by infinitely many queries that come through the output sockets afterwards.

5.3.2 Process QA-semantics

Definition 23. The *event-based QA-semantics* of a process N is defined as:

$$\llbracket \mathrm{N} \rrbracket^{\mathbf{E}}_{\mathrm{QA}} : \prod_{s^- \in S^-_{\mathrm{N}}} \left\llbracket \mathcal{P}^{s^-}_{\mathrm{N}} \right\rrbracket_{\mathrm{QA}} \to \prod_{s^+ \in S^+_{\mathrm{N}}} \left\llbracket \mathcal{P}^{s^+}_{\mathrm{N}} \right\rrbracket_{\mathrm{QA}}$$

$$\begin{aligned} &\left(\llbracket \mathrm{N} \rrbracket^{\mathbf{E}}_{\mathrm{QA}} \left((d_{s^-})_{s^- \in S^-_{\mathrm{N}}}\right)\right)_{s^+} (q_{s^+}) := \\ &\quad \bigsqcap \Big\{ \textstyle\bigsqcup_{i \in I} a^+_i \;\Big|\; \exists (\Sigma, E, \eta, \rhd\!\!\bullet) \in \mathfrak{T}_{\mathrm{N}}, (\beta^+_i, q^+_i)_{i \in I} : \\ &\qquad\qquad \forall i \in I.\ \eta(\beta^+_i) = (s^+, (q^+_i, a^+_i)),\ q^+_i \sqsupseteq q_{s^+}, \\ &\qquad\qquad \forall s^- \in S^-_{\mathrm{N}},\ J \subseteq I,\ \left(\beta^-_j, q^-_j, a^-_j\right)_{j \in J} : \\ &\qquad\qquad\quad \forall j \in J.\ \beta^+_j \rhd\!\!\bullet\ \beta^-_j,\ \eta(\beta^-_j) = (s^-, (q^-_j, a^-_j)) : \\ &\qquad\qquad\qquad\qquad d_{s^-}\left(\textstyle\bigsqcup_{j \in J} q^-_j\right) \sqsubseteq \textstyle\bigsqcup_{j \in J} a^-_j \Big\} \end{aligned}$$

i. e. the function that for a tuple of query-to-answer mappings representing a lower bound on what is available through the *input* sockets, returns a lower bound on the query-to-answer mappings the process will offer through its *output* sockets.

The reason $q^+_i \sqsupseteq q_{s^+}$ has been used in Def. 23 above instead of equality is that an element $q_{s^+} \in \llbracket \mathcal{P} \rrbracket_{\mathrm{Q}}$ need not necessarily be the semantics of a single query, even though it may well be the limit of the semantics of (infinitely many) other queries. In such cases equality would lead to an infimum over the empty set, which is $\top$.

As before, the trace-based semantics is much simpler to define:

Definition 24. The *trace-based QA-semantics* of a process N is defined as:

$$\llbracket \mathrm{N} \rrbracket^{\mathbf{T}}_{\mathrm{QA}} \left((d_{s^-})_{s^- \in S^-_{\mathrm{N}}}\right) := \bigsqcap \left\{ \left(\llbracket \zeta_{s^+} \rrbracket_{\mathrm{QA}}\right)_{s^+ \in S^+_{\mathrm{N}}} \;\middle|\; \zeta \in \mathfrak{T}_{\mathrm{N}}, \forall s^- \in S^-_{\mathrm{N}} : \llbracket \zeta_{s^-} \rrbracket_{\mathrm{QA}} \sqsupseteq d_{s^-} \right\}$$

Having no hidden state guarantees the equality of the two variants of QA-semantics, in much the same way it did for Q-semantics in Lemma 22:

Lemma 25. *For any process* N *without hidden state, we have* $\llbracket \mathrm{N} \rrbracket^{\mathbf{T}}_{\mathrm{QA}} = \llbracket \mathrm{N} \rrbracket^{\mathbf{E}}_{\mathrm{QA}}$.

While the proof of this lemma is bound to be more technically complex than it was for Q-semantics, its essence is the same: each trace can be represented by its groups of causally-closed query-answer pairs and each output-input causality can be isolated in a separate trace. We therefore omit a detailed proof.

For a process N without hidden state we denote its QA-semantics simply as $\llbracket \mathrm{N} \rrbracket_{\mathrm{QA}}$.

5.3.2.1 Semantics of example process

The multiplication process given in Fig. 3 has fairly simple Q- and QA-semantics. The Q-semantics is $\lambda i.(i, i)$. Such semantics is interpreted as follows: whenever the output socket has only queries below or equal i, then each of the input sockets will be given only queries below or equal i. Estimates of queries in this direction are useful for giving a condition that prevents queries from forming infinite dependency cycles in networks (see [Konečný and Farjudian 2010]).

The QA-semantics of the multiplication process is the map that takes a pair of mappings of type $[\omega_\perp^\top \to \mathbb{IR}^\top]$ to a mapping of the same type by point-wise multiplying the two mappings. Thanks to the consideration of queries that are not directly expressible in the protocol, i. e. the "query" $\top$ in this case, the QA-semantics contains the information that the process does *not* give partial or incorrect answers when multiplying two exact numbers in the limit.

5.3.3 Process responsiveness and consistency

Unfortunately, one cannot deduce from a process' QA-semantics that the process always gives correct answers. For example, even when the QA-semantics says that a network computes the exact product of its inputs, it may in fact be the case that the network rarely answers any query sent to it. This is because QA-semantics has no way of capturing whether or not queries always get answered and whether or not answers exclude $\top$. When taking alternative traces into account and some of them map some queries to $\top$ while others don't, the infimum in Def. 24 will hide the occurrence of $\top$.

If we can show that a network answers all its queries and the answers are not $\top$, we have ruled out that the QA-semantics could be $\top$ unless certain queries are genuinely never asked. It is therefore important to find ways to prove that a process answers all its queries and it does so in a consistent manner. We start by defining what this means.

Definition 26. A process N is called *responsive* iff there is no trace $\zeta \in \mathfrak{T}_{\mathrm{N}}$ in which there is an unanswered query β on an output socket while all queries in ζ that are directly caused by β are answered.

Definition 27. A process N is called *consistent* iff there is no trace $\zeta \in \mathfrak{T}_{\mathrm{N}}$ in which $[\![\zeta_{s^+}]\!]_{\mathrm{A}} = \top$ for some output socket s^+ while $[\![\zeta_{s^-}]\!]_{\mathrm{A}} \neq \top$ for all input sockets s^-.

Since all processes we consider are responsive, we will focus on maintaining this property via composition (see Part II). To show the consistency of a composite process, we will have to develop other means because one important basic process is not consistent, namely N^{merge} — unless the two input sockets of N^{merge} give consistent values, the output is not consistent. To handle such processes, we introduce the following property:

Definition 28. A process N is called *QA-consistent* if for each trace $\zeta \in \mathfrak{T}_{\mathrm{N}}$ and output socket s^+

$$[\![\mathrm{N}]\!]_{\mathrm{QA}}\left(\left([\![\zeta_{s^-}]\!]_{\mathrm{QA}}\right)_{s^-\in S^-}\right)([\![\zeta_{s^+}]\!]_{\mathrm{Q}}) = [\![\zeta_{s^+}]\!]_{\mathrm{A}}$$

By definition of QA-semantics, we always have the $\sqsubseteq$ part of the above equality. In a QA-consistent process all traces agree about the result in the limit, and thus avoid weakening the process QA-semantics. In other words, a QA-consistent process is almost extensional, i. e. despite various differences in execution due to the representation of input information and nondeterminism, whenever the inputs have the same QA-semantics, the output is unambiguous in its limit, including the presence or absence of inconsistencies.

All processes we have considered so far are QA-consistent. Some important processes that are not easy to make QA-consistent could be ones that require many-valuedness. For example, a process that given coefficients of a third-order polynomial, returns one of its real roots cannot be extensional, i. e. for different representations of the same coefficients, it sometimes returns a different root. We do not deal with such processes in this article but plan to support them using appropriate power-domain semantics and thus maintaining a weaker version of QA-consistency.

5.3.4 Input-relative Q-semantics

Consider a process that computes the identity function on real numbers using a protocol whose queries are natural numbers and answers are rational intervals. Additionally the process guarantees the convergence rate $\lambda n.2^{-n}$ on its output socket. The standard Q-semantics will be constant $\top$ as one cannot put an upper bound on the size of queries that will have to be made to get even the first approximation of the required size. To have a useful Q-semantics for such cases, we define a Q-semantics relative to a certain lower bound QA-semantics on the input channels:

Definition 29 (Input specific Q-semantics). The *Q-semantics* of a process N without hidden state *relative to input* QA-semantics $(d_{s^-})_{s^-\in S_N^-}$ is defined as:

$$\left[\!\left[\mathrm{N}/(d_{s^-})_{s^-\in S_\mathrm{N}^-}\right]\!\right]_\mathrm{Q}\left((q_{s^+})_{s^+\in S_\mathrm{N}^+}\right) := \\ \bigsqcup\left\{\left([\![\zeta_{s^-}]\!]_\mathrm{Q}\right)_{s^-\in S_\mathrm{N}^-} \;\middle|\; \zeta\in\mathfrak{T}_\mathrm{N}, \forall s^+\in S_\mathrm{N}^+ : [\![\zeta_{s^+}]\!]_\mathrm{Q}\sqsubseteq q_{s^+}, \forall s^-\in S_\mathrm{N}^- : [\![\zeta_{s^-}]\!]_\mathrm{QA}\sqsupseteq d_{s^-}\right\}$$

i. e. the function that given an upper bound on what has been asked on the *output* sockets, for each of its input sockets s^-, returns the least upper bound on what the process consequently asks through this input socket, assuming the input's QA-semantics is above the given bound $(d_{s^-})_{s^-\in S_N^-}$.

In our example exponential rate identity process, if is is known that the input is provided with QA-semantics that implies the convergence rate $\lambda n.2^{1-n}$, the process has the Q-semantics $\lambda q.q + 1$ specific to this input.

5.3.5 Process continuity

Lemma 30. $[\![\mathrm{N}]\!]^\mathbf{E}_\mathrm{Q}$, $[\![\mathrm{N}]\!]^\mathbf{T}_\mathrm{Q}$, $[\![\mathrm{N}]\!]^\mathbf{E}_\mathrm{QA}$ *and* $[\![\mathrm{N}]\!]^\mathbf{T}_\mathrm{QA}$ *are* monotone *for any process* N.

Proof. Increasing the argument increases (resp. decreases) the set of traces considered, which increases the supremum (resp. infimum) of the semantics of these traces. Also with increasing the argument, the set of events considered increases (resp. decreases), which has the same monotone effect. □

In light of Lemma 30 one may expect to be able to extend the result to continuity. However, as we have not considered any language or any restrictive method of defining processes, they may well be non-computable with non-continuous behaviour. Moreover, even if the processes are computable and thus have a continuous low-level trace semantics based on time-ordered sequences, our time-agnostic semantics is not necessarily continuous for non-maximal elements. Next, we discuss such an example.

5.3.5.1 Non-continuous process example

An example non-continuous process is one that has one input and one output socket, both using the $\mathcal{P}_{\mathrm{Fn}}$ protocol, and answers some queries directly with $\perp$, while it forwards other queries to its input socket. When forwarding a query sometimes it forwards the answer it receives and sometimes it forgets the answer and answers $\perp$ instead.

Assume all queries strictly below $([0,1],2)$ (such as $([0,1/2],2)$ and $([0,1],1)$) are ignored while all others, including $([0,1],2)$ itself are forwarded. Then the Q-semantics maps $([0,1],2)$ to itself but all strictly lower queries it maps to $\perp$. This means that Q-semantics is not continuous as the supremum of all queries strictly below $([0,1],2)$ is $([0,1],2)$ but the supremum of their Q-semantics is $\perp$, which is different from the Q-semantics of $([0,1],2)$.

Also assume that all answers strictly below $[0,1]\searrow[0,1]$ (such as $[0,1]\searrow[0,2]$ or $[0,1/2]\searrow[0,1]$) are replaced by $\perp$ whereas all others are forwarded. The QA-semantics maps $\lambda q.([0,1]\searrow[0,1])$ to itself whereas the functions $\lambda q.d$ for any d strictly below $[0,1]\searrow[0,1]$ are all mapped to $\perp$. As these functions converge to $\lambda q.([0,1]\searrow[0,1])$ but their QA-semantics do not converge to $\lambda q.([0,1]\searrow[0,1])$, the QA-semantics is not continuous.

6 Summary and future work

We have laid the foundations of a framework in which one can study *distributed* query-answer based computation. We have presented the basic definitions up to the process level, providing, in particular, 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;

We have had in mind the *exact* computation over real numbers, and other similar continuous types — on ground or higher orders — as encountered in analysis and geometry.

In related work [Konečný and Farjudian 2010] we provide basic definitions of finite or infinite networks built over these processes and establish results on how to *safely estimate the semantic behaviour of a finite or infinite process network from the behaviour of its components*.

We also plan to make use of our explicit effort indicators to measure communication, time and space complexity of processes using query-driven communication of exact values.

References

[Amadio and Curien 1998] Amadio, R. and Curien, P.-L.: *Domains and Lambda Calculi*, volume 46 of *Cambridge Tracts in Theoretical Computer Science* Cambridge University Press, 1998.

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

[Abramsky et al. 2000] Abramsky, S., Jagadeesan, R., and Malacaria, P.: Full abstraction for PCF; *Information and Computation*, 163(2):409–470, 2000.

[Edalat and Potts 2000] Edalat, A. and Potts, P. J.: A new representation for exact real numbers; In Brookes, S. and Mislove, M., editors, *Electronic Notes in Theoretical Computer Science*, volume 6, pages 119–132. Elsevier Science Publishers, 2000.

[Escardó 1997] Escardó, M. H.: *PCF extended with real numbers: a domain theoretic approach to higher order exact real number computation* PhD thesis, Imperial College, 1997.

[Gierz et al. 2003] Gierz, G., Hofmann, K. H., Keimel, K., Lawson, J. D., Mislove, M. W., and Scott, D. S.: *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications* Cambridge University Press, 2003.

[Gosper 1972] Gosper, R. W.: Continued fraction arithmetic; Technical Report HAKMEM Item 101B, MIT AI MEMO 239, MIT, February 1972.

[Heckmann 1998] Heckmann, R.: Big integers and complexity issues in exact real arithmetic; In Edalat, A., Jung, A., Keimel, K., and Kwiatkowska, M., editors, *Comprox III, Third Workshop on Computation and Approximation*, volume 13 of *Electr. Notes Theor. Comput. Sci.*, page 69. Elsevier, 1998.

[Kahn 1974] Kahn, G.: The semantics of a simple language for parallel programming; In Rosenfeld, J. L., editor, *Information Processing, Proc. of IFIP, Stockholm, August 5-10, 1974*, pages 471–475. North Holland, 1974.

[Konečný and Farjudian 2010] Konečný, M. and Farjudian, A.: Compositional semantics of dataflow networks with query-driven communication of exact values; *Journal of Universal Computer Science*, 16(18):2629–2656, 2010.

[Konečný 2008] 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.

[Lambov 2007] Lambov, B.: Reallib: An efficient implementation of exact real arithmetic; *Mathematical Structures in Computer Science*, 17(1):81–98, 2007.

[Moore 1966] Moore, R. E.: *Interval Analysis* Prentice-Hall, Englewood Cliffs, NJ, USA, 1966.

[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.

[Neumaier 2003] Neumaier, A.: Taylor forms - use and limits; *Reliable Computing*, 9(1):43–79, 2003.

[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.

[Revol et al. 2005] Revol, N., Makino, K., and Berz, M.: Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY; *Journal of Logic and Algebraic Programming*, 64:135–154, 2005.

[Turing 1935] Turing, A. M.: On computable numbers with an application to the Entscheidungsproblem; *Proceedings of London Mathematical Society. II. Series.*, 42:230–265, 1935.

[Weihrauch 1987] Weihrauch, K.: *Computability* Number 9 in EATCS Monographs on Theoretical Computer Science. Springer, Berlin, 1987.

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