A Domain-Specific Language for Incremental and Modular Design of Large-Scale Verifiably-Safe Flow Networks (Preliminary Report)

Azer Bestavros
Boston University
best@bu.edu

Assaf Kfoury
Boston University
kfoury@bu.edu

We define a *domain-specific language* (DSL) to inductively assemble *flow networks* from *small networks* or *modules* to produce arbitrarily large ones, with interchangeable functionally-equivalent parts. Our small networks or modules are "small" only as the building blocks in this inductive definition (there is no limit on their size). Associated with our DSL is a *type theory*, a system of formal annotations to express desirable properties of flow networks together with rules that enforce them as *invariants* across their interfaces, *i.e.*, the rules guarantee the properties are preserved as we build larger networks from smaller ones. A prerequisite for a type theory is a *formal semantics*, *i.e.*, a rigorous definition of the entities that qualify as feasible flows through the networks, possibly restricted to satisfy additional efficiency or safety requirements. This can be carried out in one of two ways, as a *denotational* semantics or as an *operational* (or *reduction*) semantics; we choose the first in preference to the second, partly to avoid exponential-growth rewriting in the operational approach. We set up a typing system and prove its soundness for our DSL.

1 Introduction and Motivation

Flow Networks. Most large-scale systems can be viewed as assemblies of subsystems, or gadgets, each of which produces, consumes, or regulates a flow of some sort. In a computer network, a discrete flow of messages (packets) is produced by servers (*e.g.*, streaming sources), regulated by network devices (*e.g.*, routers and shapers), and consumed by clients (*e.g.*, stream players). In a road network, the flow constitutes vehicles which enter and exit at edge exchanges, and which are regulated by speed limits on road segments, and by traffic lights at forks and intersections. In electric grids, a continuous flow of energy (electric current flow) is produced by power sources, regulated by transformers, transported by transmission lines, and consumed by power sinks. In a sensor network, a flow of measurements is produced by sensors, regulated by filters and repeaters, and consumed by sinks and aggregators. In a computing grid or cloud, a flow of resources (*e.g.*, CPU cycles) is produced by physical clusters of hosts, regulated by schedulers, resource managers, and hypervisors, and consumed by applications.

In each of the above systems, a "network" is assembled from smaller building blocks, which themselves could be smaller, inductively assembled networks or alternately, they could be individual *modules*. Thus, what we call *flow networks* are inductively defined as assemblies of *small networks* or *modules*. The operation of a flow network is characterized by a set of variables and a set of constraints thereof, reflecting *basic*, *assumed*, or *inferred* properties or rules governing how the network operates, and what constitutes safe operation. Basic rules (variables and constraints) are inherently defined, and are typically specified by a domain expert for individual modules. Assumed rules are speculatively specified for outsourced or yet-to-be fleshed out networks, which constitute *holes* in a larger network. Holes in a network specification allow the design or analysis of a system to proceed based only on promised functionality of missing modules or networks to be plugged in later. Inferred rules are those that could be

Submitted to:
DSL 2011

© A. Bestavros & A. Kfoury
This work is licensed under the
Creative Commons Attribution License.

derived through repeated composition and analysis of networks. Such derivations may be exact, or may underscore conservative approximations (*e.g.*, upper or lower bounds on variables or expressions).

Basic or inferred rules – underscoring constraints on the operation of a flow network – could be the result of analysis using any one of a set of diverse theories or calculi. For instance, in a streaming network application, the size of a maximum burst of packets produced by a server over a window of time may be bounded using analysis that relies on real-time scheduling theory, whereas the maximum burst of packets emitted by a sequence of networking elements (*e.g.*, multicast routers and shapers) over a (possibly different) window of time may be bounded using analysis that relies on network calculus [7]. Clearly, when a larger flow network consisting of streaming servers as well as network elements – not to mention holes – is assembled, neither of these underlying calculi on its own could be used to perform the requisite network-wide analysis to derive the rules at the boundaries of the larger flow network. Rather, the properties at the boundaries of the constituent (smaller) networks of servers and networking elements constitute a domain-specific language (of maximum burst size over time, in this case), the semantics of which can be used to derive the rules at the boundaries of the larger flow network.

Several approaches to system design, modeling and analysis have been proposed in recent years, overlapping with our notion of flow networks. Apart from the differences in the technical details – at the level of formalisms and mathematics that are brought to bear – our approach distinguishes itself from the others by incorporating from its inception three inter-related features/goals: (a) the ability to pursue system design and analysis without having to wait for missing (or broken) components/modules to be inserted (or replaced), (b) the ability to abstract away details through the retention of only the salient variables and constraints at network interfaces as we transition from smaller to larger networks, and (c) the ability to leverage diverse, unrelated theories to derive properties of modules and small networks, as long as such networks share a common formal language at their interfaces – a formal Domain-Specific Language (DSL) that enables assembly and analysis that is agnostic to the underlying theory used to derive such properties.

Examples of DSL Use Cases. Before delving into the precise definitions and formal arguments of our DSL, we provide brief descriptions of how flow networks could be leveraged for two application domains – namely resource allocation and arbitration subject to Service Level Agreements (SLAs) for *video streaming* in a *cloud computing* setting, and emerging safety-critical CPS and *smart grid* applications.

The generality of our DSL is such that it can be applied to problems in settings that are not immediately apparent as flow network settings. For example, consider a single, physical or virtual host (processor). One may view such a host i as the source of a *supply flow* of compute cycles, offered in constant increments c_i every period t_i. Similarily, a process or application j executing on such a host can be viewed as a *demand flow* of compute cycles, requested periodically with some characteristics – *e.g.*, subject to a maximum consumption of w_j cycles per period t_j. In this setting, multiple supply flows (*e.g.* a set of processors in a multicore/cluster setting), each represented by an individual supply (c_i,t_i) flow, can be regulated/managed using hypervisor system software to yield a flow network that exhibits a more elaborate pattern of compute cycles. For instance, the resulting flow may be specified as a single (c_m,t_m) flow, where c_m cycles are supplied over the Least Common Multiple (LCM) period t_m, or it may be sepcified as a set of (c_k,t_k) flows, each of which operating at some discrete period t_k drawn from the lattice of LCM periods defined by the individual t_i periods. Similarily, multiple demand flows (*e.g.* a set of services offered within a single virtual machine), each represented by an individual demand (w_j,t_j) flow, can be multiplexed to yield more elaborate consumption patterns of the resulting workload. Finally, a supply flow may be matched up to a set of demand flows through the use of a scheduler. Clearly, for a flow network of compute cycle producers, consumers, and schedulers to operate safely, specific constraints (rules) must be satisfied. For instance, matching up supply and demand flows adhere to a “supply meets demand” condition, or to some other SLA, such as “periods of overload cannot exceed 100 msecs”

or "no more than 5 missed periodic allocations in any 1-minute window of time".

Not only is our DSL useful in modeling the supply of, demand for, and consumption (through a scheduler) of compute cycles, but also in a very similar manner they can be used readily to model the supply of, demand for, and consumption (through resource management protocols) of other computing resources such as network bandwidth, storage capacities, *etc.*

In the above setting, the flow networks describing the supply, demand, or scheduling of computing and networking resources can be made as small as desired to render their whole-system analysis tractable, or as large as desired to produce more precise system-wide typings. For instance, readers familiar with the vast literature on real-time scheduling (*e.g.*, [21, 23, 24]) will immediately recognize that most of the results in that literature can be viewed as deriving fairly tight bounds on specific processor schedulers such as EDF, RMS, Pinwheel, among others schedulers. Similarily, readers familiar with QoS provisioning using network calculus, traffic envelopes, fluid network models will recognize that most of the results obtained through these models are applicable for specific protocols such as AIMD, weighted-fair queuing, among other schedulers (*e.g.*, [7, 20, 26]).

Modeling and analysis of the supply of (and demand for) computing and networking resources is particularly valuable in the context of cloud and grid resource management (*e.g.*, [1, 8, 14, 17, 27]). In such a setting, a cloud operator may use a DSL to specify the topological configuration of computing and networking resources, the layer of system software used to virtualize these resources, as well as a particular mapping of client workloads to virtualized resources. Compiling such a DSL-sepecification is akin to verifying the safety of the system. Moreover, making changes to these DSL specifications enables the operator (or a mechanized agent thereof) to explore whether an alternative arrangement of resources or an alternative mapping of client workloads is more efficient [16].

As another example of the broad applicablity of our DSL, consider yet another application domain – that of smart electric grids. In this domain, a module would be a grid "cell", such as a power plant, a residential or commercial building, a power transmission line, a transformer, or a power storage facility (batteries), *etc.* Each cell has a capacity to produce and consume power over time (energy flow). For example, a house with solar panels may be contributing a positive flow to the grid or a negative flow depending on the balance between solar panel supply and house demand. Operational or safety constraints on cells and interconnections of cells define relationships that may be the subject of exact whole-system analysis on the small scale, or approximate compositional analysis on the large scale. The simplest of cells is perhaps a transmission line, which may be modeled by input and output voltages v_{in} and v_{out}, a maximum allowable drop in voltage δ_v, a resistance R which is a function of the medium and transmission distance, a current rating I, and a power rating P. Ignoring delays, one can describe such a cell by a set of constraints: *e.g.*, $v_{out} = v_{in} - R*I$ (the voltage at the output is the difference between the input voltage and the voltage drop due to resistance), $v_{out} * I \leq P$ (the power drain cannot exceed a maximum rated wattage), and $R*I \leq \delta_v$ (the drop in voltage must be less than what is allowed). Similarly, modules for other types of cells may be specified (or left unspecified as holes) and arrangements of such modules may be used to model large-scale smart grids, allowing designers to explore "what if" scenarios, *e.g.*, under what conditions would a hole in the grid cause a safety violation? or what are the most efficient settings (*e.g.*, power generation and routing decisions) in terms of power loss due to inefficient transmission? The introduction of "smart" computational processes in the grid (*e.g.*, feedback-based power management) and the expected diversity of technologies to be plugged into the grid make the consideration of such questions quite critical.

A Type Theory and Formal Semantics of Flow Networks. Associated with our DSL is a *type theory*, a system of formal annotations to express desirable properties of flow networks together with rules that enforce them as *invariants* across their interfaces, *i.e.*, the rules guarantee the properties are preserved as we build larger networks from smaller ones.

A prerequisite for a type theory is a *formal semantics* – a rigorous definition of the entities that qualify as feasible flows through the networks, possibly restricted to satisfy additional efficiency or safety requirements. This can be carried out in one of two ways, as a *denotational* semantics or as an *operational* (or *reduction*) semantics. In the first approach, a feasible flow through the network is denoted by a function, and the semantics of the network is the set of all such functions. In the second approach, the network is uniquely rewritten to another network in *normal form* (appropriately defined), and the semantics of the network is its normal form or directly extracted from it. Though the two can be shown to be equivalent (in a sense that can be made precise), whenever we need to invoke a network's semantics, we rely on the denotational definition in order to avoid complexity issues related to the operational definition. Some of these complexity issues are already evident from the form of network specifications we can write in our DSL.

As we alluded before, a distinctive feature of our DSL is the presence of *holes* in network specifications, together with constructs of the form: **let** $X = \mathcal{M}$ **in** $\mathcal{N}$, which informally says "network $\mathcal{M}$ may be safely placed in the occurrences of hole X in network $\mathcal{N}$". What "safely" means will later depend on the invariant properties that typings are formulated to enforce. There are other useful *hole-binders* besides **let-in**, which we denote **try-in**, **mix-in**, and **letrec-in**. An informal explanation of what these hole-binders mean is in Remark 6 and Example 7.

Rewriting a specification in order to eliminate all occurrences of holes and hole-binders is a costly process, generally resulting in an exponential growth in the size of the expression denoting the specification, which poses particular challenges in the definition of an operational semantics. We set up a typing system and prove its soundness for our DSL without having to explicitly carry out such exponential-growth rewriting.

Our DSL provides two other primitive constructs, one of the form $\big(\mathcal{M}_1 \parallel \mathcal{M}_2\big)$ and another of the form **bind** $(\mathcal{N}, \langle a, b \rangle)$. The former juxtaposes two networks $\mathcal{M}_1$ and $\mathcal{M}_2$ in parallel, and the latter binds the output arc a of a network $\mathcal{N}$ to its input arc b. With these primitive or core constructors, we can define many others as *derived* constructors and according to need.

Paper Overview and Context. The remainder of this paper is organized as follows. Section 2 is devoted to preliminary definitions. Section 3 introduces the syntax of our DSL and lays out several conditions for the well-formedness of network specifications written in it. We only include the **let-in** constructor, delaying the full treatment of **try-in**, **mix-in**, **letrec-in**, to subsequent reports.

The formal semantics of flow networks are introduced in Section 4 and a corresponding type theory is presented in Section 5. The type theory is syntax-directed, and therefore *modular*, as it infers or assigns typings to objects in a stepwise inside-out manner. If the order in which typings are inferred for the constituent parts does not matter, we additionally say that the theory is *fully compositional*. We add the qualifier "fully" to distinguish our notion of compositionality from similar, but different, notions in other areas of computer science.[1] We only include an examination of modular typing inference in this paper, leaving its (more elaborate) fully-compositional version to a follow-up report.

The balance of this paper expands on the fundamentals laid out in the first four sections: Sections 6 to 10 mostly deal with issues of typing inference, whether for the basic semantics of flow networks (introduced in Section 4) or their relativized semantics, whereby flows are feasible if they additionally satisfy appropriately defined objective functions (introduced in Section 9).

Acknowledgment. The work reported in this paper is a small fraction of a collective effort involving several people, under the umbrella of the **iBench Initiative** at Boston University. The reader is invited to visit the website `https://sites.google.com/site/ibenchbu/` for a list of participants,

[1] Adding to the imprecision of the word, "compositional" in the literature is sometimes used in the more restrictive sense of "modular" in our sense.

former participants, and other research activities. The DSL presented in this paper, with its formal semantics and type system, is in fact a specialized and simpler version of a DSL we introduced earlier in our work for NetSketch, an integrated environment for the modeling, design and analysis of large-scale safety-critical systems with interchangeable parts [5, 6, 25]. In addition to its DSL, NetSketch has two other components currently under development: an automated verifier (AV), and a user interface (UI) that combines the DSL and the AV and adds appropriate tools for convenient interactive operation.

2 Preliminary Definitions

A *small network* $\mathcal{A}$ is of the form $\mathcal{A} = (\mathbf{N}, \mathbf{A})$ where $\mathbf{N}$ is a set of nodes and $\mathbf{A}$ a set of directed arcs. Capacities on arcs are determined by a lower-bound $L : \mathbf{A} \to \mathbb{R}^+$ and an upper-bound $U : \mathbf{A} \to \mathbb{R}^+$ satisfying the conditions $L(a) \leqslant U(a)$ for every $a \in \mathbf{A}$. We write $\mathbb{R}$ and $\mathbb{R}^+$ for the sets of all reals and all non-negative reals, respectively. We identify the two ends of an arc $a \in \mathbf{A}$ by writing *head*(a) and *tail*(a), with the understanding that flow moves from *tail*(a) to *head*(a). The set $\mathbf{A}$ of arcs is the disjoint union (denoted "$\uplus$") of three sets: the set $\mathbf{A}_\#$ of internal arcs, the set $\mathbf{A}_{\text{in}}$ of input arcs, and the set $\mathbf{A}_{\text{out}}$ of output arcs:

$$
\begin{array}{lcl}
\mathbf{A} & = & \mathbf{A}_\# \uplus \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} \quad \text{where} \\
\mathbf{A}_\# & = & \{ a \in \mathbf{A} \mid \mathit{head}(a) \in \mathbf{N} \text{ and } \mathit{tail}(a) \in \mathbf{N} \} \\
\mathbf{A}_{\text{in}} & = & \{ a \in \mathbf{A} \mid \mathit{head}(a) \in \mathbf{N} \text{ and } \mathit{tail}(a) \notin \mathbf{N} \} \\
\mathbf{A}_{\text{out}} & = & \{ a \in \mathbf{A} \mid \mathit{head}(a) \notin \mathbf{N} \text{ and } \mathit{tail}(a) \in \mathbf{N} \}
\end{array}
$$

The tail of an input arc, and the head of an output arc, are not attached to any node. We do not assume $\mathcal{A}$ is connected as a directed graph – a sensible assumption in studies of network flows, whenever there is only one input arc (or "source node") and one output arc (or "sink node"). We assume $\mathbf{N} \neq \varnothing$, *i.e.*, there is at least one node in $\mathbf{N}$, without which there would be no input and no output arc, and nothing to say.

A *flow* f in $\mathcal{A}$ is a function that assigns a non-negative real to every $a \in \mathbf{A}$. Formally, a flow is a function $f : \mathbf{A} \to \mathbb{R}^+$ which, if *feasible*, satisfies "flow conservation" and "capacity constraints" (below).

We call a bounded interval $[r, r']$ of reals, possibly negative, a *type*, and we call a *typing* a function T that assigns a type to every subset of input and output arcs. Formally, T is of the following form:[2]

$$T \ : \ \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \ \to \ \mathbb{R} \times \mathbb{R}$$

where $\mathscr{P}(\ )$ is the power-set operator, *i.e.*, $\mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) = \{A \mid A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}\}$. As a function, T is not totally arbitrary and satisfies certain conditions, discussed in Section 5, which qualify it as a *network typing*. Instead of writing $T(A) = \langle r, r' \rangle$, where $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, we write $T(A) = [r, r']$. We do not disallow the possibility that $r > r'$ which will be an empty type satisfied by no flow.

Informally, a typing T imposes restrictions on a flow f relative to every $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$ which, if satisfied, will guarantee that f is feasible. Specifically, if $T(A) = [r, r']$, then T requires that the part of f entering through the arcs in $A \cap \mathbf{A}_{\text{in}}$ minus the part of f exiting through the arcs in $A \cap \mathbf{A}_{\text{out}}$ must be within the interval $[r, r']$.

Remark 1. Let $\mathcal{A} = (\mathbf{N}, \mathbf{A})$ be a small network. We may want to identify some nodes as *producers* and some others as *consumers*. In the presence of lower-bound and upper-bound functions L and U, we do not need to do this explicitly. For example, if n is a node that produces an amount $r \in \mathbb{R}^+$, we introduce

[2] Our notion of a "typing" as an assignment of types to the members of a powerset is different from a similarly-named notion in the study of type systems for programming languages. In the latter, a typing refers to a derivable "typing judgment" consisting of a program expression M, a type assigned to M, *and* a type environment with a type for every free variable in M.

instead a new input arc a entering n with $L(a) = U(a) = r$. Similarly, if n' is a node that consumes an amount $r' \in \mathbb{R}^+$, we introduce a new output arc a' exiting n' with $L(a') = U(a') = r'$. The resulting network $\mathcal{A}'$ is equivalent to $\mathcal{A}$, in that any feasible flow in $\mathcal{A}'$ induces a feasible flow in $\mathcal{A}$, and vice-versa. □

Flow Conservation, Capacity Constraints, Type Satisfaction. Though obvious, we precisely state fundamental concepts underlying our entire examination and introduce some of our notational conventions, in Definitions 2, 3, 4, and 5.

Definition 2 (*Flow Conservation*)**.** If A is a subset of arcs in $\mathcal{A}$ and f a flow in $\mathcal{A}$, we write $\sum f(A)$ to denote the sum of the flows assigned to all the arcs in A: $\sum f(A) = \sum\{f(a) \,|\, a \in A\}$. By convention, $\sum\varnothing = 0$. If $A = \{a_1,\dots,a_p\}$ is the set of all arcs entering node n, and $B = \{b_1,\dots,b_q\}$ is the set of all arcs exiting node n, then conservation of flow at n is expressed by the linear equation:

(1) $\sum f(A) = \sum f(B)$

There is one such equation for every node $n \in \mathbf{N}$. □

Definition 3 (*Capacity Constraints*)**.** A flow f satisfies the capacity constraints at arc $a \in \mathbf{A}$ if:

(2) $L(a) \leqslant f(a) \leqslant U(a)$

There are two such inequalities for every arc $a \in \mathbf{A}$. □

Definition 4 (*Feasible Flows*)**.** A flow f is *feasible* iff two conditions:

- for every node $n \in \mathbf{N}$, the equation in (1) is satisfied,
- for every arc $a \in \mathbf{A}$, the two inequalities in (2) are satisfied,

following standard definitions of network flows. □

Definition 5 (*Type Satisfaction*)**.** Let $T : \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \to \mathbb{R} \times \mathbb{R}$ be a typing for the small network $\mathcal{A}$. We say the flow f *satisfies* T if, for every $A \in \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$ with $T(A) = [r, r']$, it is the case:

(3) $r \leqslant \sum f(A \cap \mathbf{A}_{\text{in}}) - \sum f(A \cap \mathbf{A}_{\text{out}}) \leqslant r'$

We often denote a typing T for $\mathcal{A}$ by simply writing $\mathcal{A} : T$. □

3 DSL for Incremental and Modular Design of Flow Networks (Untyped)

The definition of small networks in Section 2 was less general than our full definition of networks, but it had the advantage of being more directly comparable with standard graph-theoretic definitions. Our networks in general involve what we call "holes". A *hole* X is a pair $(\mathbf{A}_{\text{in}}, \mathbf{A}_{\text{out}})$ where $\mathbf{A}_{\text{in}}$ and $\mathbf{A}_{\text{out}}$ are disjoint finite sets of input and output arcs. A hole X is a place holder where networks can be inserted, provided the *matching-dimensions* condition (in Section 3.2) is satisfied.

We use a BNF definition to generate formal expressions, each being a formal description of a network. Such a formal expression may involve subexpressions of the form: **let** $X = \mathcal{M}$ **in** $\mathcal{N}$, which informally says "$\mathcal{M}$ may be safely placed in the occurrences of hole X in $\mathcal{N}$". What "safely" means depends on the invariant properties that typings are formulated to enforce. In such an expression, we call the X to the left of "=" a *binding* occurrence, and we call all the X's in $\mathcal{N}$ *bound* occurrences.

If $\mathcal{A} = (\mathbf{N}, \mathbf{A})$ is a small network where $\mathbf{A} = \mathbf{A}_{\#} \uplus \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$, let $\mathbf{in}(\mathcal{A}) = \mathbf{A}_{\text{in}}$, $\mathbf{out}(\mathcal{A}) = \mathbf{A}_{\text{out}}$, and $\#(\mathcal{A}) = \mathbf{A}_{\#}$. Similarly, if $X = (\mathbf{A}_{\text{in}}, \mathbf{A}_{\text{out}})$ is a hole, let $\mathbf{in}(X) = \mathbf{A}_{\text{in}}$, $\mathbf{out}(X) = \mathbf{A}_{\text{out}}$, and $\#(X) = \varnothing$. We

assume the arc names of small networks and holes are all pairwise disjoint, *i.e.*, every small network and every hole has its own private set of arc names.

The formal expressions generated by our BNF are built up from: the set of names for small networks and the set of names for holes, using the constructors $\parallel$, **let-in**, and **bind**:

$$
\begin{array}{llll}
\mathcal{A},\mathcal{B},\mathcal{C} \in \textsc{SmallNetworks} & & & \\
X,Y,Z \in \textsc{HoleNames} & & & \\
\mathcal{M},\mathcal{N},\mathcal{P} \in \textsc{Networks} & ::= & \mathcal{A} & \text{small network name} \\
& \mid & X & \text{hole name} \\
& \mid & \mathcal{M} \parallel \mathcal{N} & \text{parallel connection} \\
& \mid & \textbf{let}\ X = \mathcal{M}\ \textbf{in}\ \mathcal{N} & \text{let-binding of hole } X \\
& \mid & \textbf{bind}\ (\mathcal{N}, \langle a,b\rangle) & \text{bind } \textit{head}(a) \text{ to } \textit{tail}(b)\text{, where} \\
& & & \langle a,b\rangle \in \textbf{out}(\mathcal{N}) \times \textbf{in}(\mathcal{N})
\end{array}
$$

where $\textbf{in}(\mathcal{N})$ and $\textbf{out}(\mathcal{N})$ are the input and output arcs of $\mathcal{N}$. In the full report [19], we formally define $\textbf{in}(\mathcal{N})$ and $\textbf{out}(\mathcal{N})$, as well as the set $\#(\mathcal{N})$ of internal arcs of $\mathcal{N}$, by structural induction.

We say a flow network $\mathcal{N}$ is *closed* if every hole X in $\mathcal{N}$ is bound. We say $\mathcal{N}$ is *totally closed* if it is closed and $\textbf{in}(\mathcal{N}) = \textbf{out}(\mathcal{N}) = \varnothing$, *i.e.*, $\mathcal{N}$ has no input arcs and no output arcs.

3.1 Derived Constructors

From the three primitive constructors introduced above: $\parallel$, **let-in**, and **bind**, we can define several other constructors. Below, we present four of these derived constructors precisely, and mention several others in Remark 6. Our four derived constructors are used as in the following expressions, where $\mathcal{N}$, $\mathcal{N}_i$, and $\mathcal{M}_j$, are network specifications and θ is set of arc pairs:

$$\textbf{bind}\ (\mathcal{N},\theta) \qquad \textbf{conn}(\mathcal{N}_1,\mathcal{N}_2,\theta) \qquad \mathcal{N}_1 \oplus \mathcal{N}_2 \qquad \textbf{let}\ X \in \{\mathcal{M}_1,\ldots,\mathcal{M}_n\}\ \textbf{in}\ \mathcal{N}$$

The second above depends on the first, the third on the second, and the fourth is independent of the three preceding it. Let $\mathcal{N}$ be a network specification. We write $\theta \subseteq_{1\text{-}1} \textbf{out}(\mathcal{N}) \times \textbf{in}(\mathcal{N})$ to denote a partial one-one map from $\textbf{out}(\mathcal{N})$ to $\textbf{in}(\mathcal{N})$. We may write the entries in θ explicitly, as in:

$$\theta = \{\langle a_1,b_1\rangle,\ldots,\langle a_k,b_k\rangle\}$$

where $a_1,\ldots,a_k \in \textbf{out}(\mathcal{N})$ and $b_1,\ldots,b_k \in \textbf{in}(\mathcal{N})$.

Our first derived constructor is a generalization of **bind** and uses the same name. In this generalization of **bind** the second argument is now θ as above rather than a single pair $\langle a,b\rangle \in \textbf{out}(\mathcal{N}) \times \textbf{in}(\mathcal{N})$. The expression $\textbf{bind}\ (\mathcal{N},\theta)$ can be expanded as follows:

$$\textbf{bind}\ (\mathcal{N},\theta) \implies \textbf{bind}\ (\textbf{bind}\ (\cdots \textbf{bind}\ (\mathcal{N},\langle a_k,b_k\rangle) \cdots, \langle a_2,b_2\rangle),\langle a_1,b_1\rangle)$$

where we first connect the head of a_k to the tail of b_k and lastly connect the head of a_1 to the tail of b_1. A little proof shows that the order in which we connect arc heads to arc tails does not matter as far as our formal semantics and typing theory is concerned.

Our second derived constructor, called **conn** (for "connect"), uses the preceding generalization of **bind** together with the constructor $\parallel$. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be network specifications, and $\theta \subseteq_{1\text{-}1} \textbf{out}(\mathcal{N}_1) \times \textbf{in}(\mathcal{N}_2)$. We expand the expression $\textbf{conn}(\mathcal{N}_1,\mathcal{N}_2,\theta)$ as follows:

$$\textbf{conn}(\mathcal{N}_1,\mathcal{N}_2,\theta) \implies \textbf{bind}\ ((\mathcal{N}_1 \parallel \mathcal{N}_2),\theta)$$

In words, **conn** connects some of the output arcs in $\mathcal{N}_1$ with as many input arcs in $\mathcal{N}_2$.

Our third derived constructor is a special case of the preceding **conn**. Unless otherwise stated, we will assume there is a fixed ordering of the input arcs and another fixed ordering of the output arcs of a network. Let $\mathcal{N}_1$ be a network specification where the number $m \geqslant 1$ of output arcs is exactly the number of input arcs in another network specification $\mathcal{N}_2$, say:

$$\mathbf{out}(\mathcal{N}_1) = \{a_1,\dots,a_m\} \quad \text{and} \quad \mathbf{in}(\mathcal{N}_2) = \{b_1,\dots,b_m\}$$

where the entries in $\mathbf{out}(\mathcal{N}_1)$ and in $\mathbf{in}(\mathcal{N}_2)$ are listed, from left to right, in their assumed ordering. Let

$$\theta = \{\langle a_1,b_1\rangle,\dots,\langle a_m,b_m\rangle\} = \mathbf{out}(\mathcal{N}_1)\times\mathbf{in}(\mathcal{N}_2)$$

i.e., the first output arc a_1 of $\mathcal{N}_1$ is connected to the first input arc b_1 of $\mathcal{N}_2$, the second output arc a_2 of $\mathcal{N}_1$ to the second input arc b_2 of $\mathcal{N}_2$, etc. Our derived constructor $(\mathcal{N}_1 \oplus \mathcal{N}_2)$ can be expanded as follows:

$$(\mathcal{N}_1 \oplus \mathcal{N}_2) \implies \textbf{conn}(\mathcal{N}_1,\mathcal{N}_2,\theta)$$

which implies that $\mathbf{in}(\mathcal{N}_1 \oplus \mathcal{N}_2) = \mathbf{in}(\mathcal{N}_1)$ and $\mathbf{out}(\mathcal{N}_1 \oplus \mathcal{N}_2) = \mathbf{out}(\mathcal{N}_2)$. As expected, $\oplus$ is associative as far as our formal semantics and typing theory are concerned, *i.e.*, the semantics and typings for $\mathcal{N}_1 \oplus (\mathcal{N}_2 \oplus \mathcal{N}_3)$ and $(\mathcal{N}_1 \oplus \mathcal{N}_2) \oplus \mathcal{N}_3$ are the same.

A fourth derived constructor generalizes **let-in** and is expanded into several nested **let**-bindings:

$$\big(\textbf{let}\ X \in \{\mathcal{M}_1,\dots,\mathcal{M}_n\}\ \textbf{in}\ \mathcal{N}\big) \implies \Big(\textbf{let}\ X_1 = \mathcal{M}_1\ \textbf{in}\ \big(\cdots\big(\textbf{let}\ X_n = \mathcal{M}_n\ \textbf{in}\ (\mathcal{N}_1 \parallel \cdots \parallel \mathcal{N}_n)\big)\cdots\big)\Big)$$

where $X_1,\dots,X_n$ are fresh hole names and $\mathcal{N}_i$ is $\mathcal{N}$ with X_i substituted for X, for every $1 \leqslant i \leqslant n$. Informally, this constructor says that *every one* of the networks $\{\mathcal{M}_1,\dots,\mathcal{M}_n\}$ can be "safely" placed in the occurrences of X in $\mathcal{N}$.

Remark 6. Other derived constructors can be defined according to need in applications. We sketch a few. An obvious generalization of $\oplus$ cascades the same network $\mathcal{N}$ some $n \geqslant 1$ times, for which we write $\bigoplus(\mathcal{N},n)$. A condition for well-formedness is that $\mathcal{N}$'s input and output dimensions must be equal.

Another derived constructor is $\textbf{Merge}(\mathcal{N}_1,\mathcal{N}_2,\mathcal{N}_3)$ which connects all the output arcs of $\mathcal{N}_1$ and $\mathcal{N}_2$ to all the input arcs of $\mathcal{N}_3$. For well-formedness, this requires the output dimensions of $\mathcal{N}_1$ and $\mathcal{N}_2$ to add up to the input dimension of $\mathcal{N}_3$. And similarly for a derived constructor of the form $\textbf{Fork}(\mathcal{N}_1,\mathcal{N}_2,\mathcal{N}_3)$ which connects all the output arcs of $\mathcal{N}_1$ to all the input arcs of $\mathcal{N}_2$ and $\mathcal{N}_3$.

While all of the preceding derived constructors can be expanded using our primitive constructors, not every constructor we may devise can be so expanded. For example, a constructor of the form

$$\textbf{try}\ X \in \{\mathcal{M}_1,\dots,\mathcal{M}_n\}\ \textbf{in}\ \mathcal{N}$$

which we can take to mean that *at least one* $\mathcal{M}_i$ can be "safely" placed in all the occurrences of X in $\mathcal{N}$, cannot be expanded using our primitives and the way we define their semantics in Section 4. Another constructor also requiring a more developed examination is of the form

$$\textbf{mix}\ X \in \{\mathcal{M}_1,\dots,\mathcal{M}_n\}\ \textbf{in}\ \mathcal{N}$$

which we can take to mean that every combination (or mixture) of *one or more* $\mathcal{M}_i$ can be selected at the same time and "safely" placed in the occurrences of X in $\mathcal{N}$, generally placing different $\mathcal{M}_i$ in different occurrences. The constructors **try-in** and **mix-in** are examined in a follow-up report. An informal understanding of how they differ from the constructor **let-in** can be gleaned from Example 7.

Another useful constructor introduces recursively defined components with (unbounded) repeated patterns. In its simplest form, it can be written as:

$$\textbf{letrec}\ X = \mathcal{M}[X]\ \textbf{in}\ \mathcal{N}[X]$$

where we write $\mathcal{M}[X]$ to indicate that X occurs free in $\mathcal{M}$, and similarly in $\mathcal{N}$. Informally, this construction corresponds to placing an open-ended network of the form $\mathcal{M}[\mathcal{M}[\mathcal{M}[\cdots]]]$ in the occurrences of X in $\mathcal{N}$. A well-formedness condition here is that the input and output dimensions of $\mathcal{M}$ must match those of X . We leave for future examination the semantics and typing of **letrec-in**, which are still more involved than those of **try-in** and **mix-in**. □

3.2 Well-Formed Network Specifications

In the full report [19], we spell out 3 conditions, not enforced by the BNF definition at the beginning of Section 3, which guarantee what we call the *well-formedness* of network specifications. We call them:

- the *matching-dimensions* condition,
- the *unique arc-naming* condition,
- the *one binding-occurrence* condition.

These three conditions are automatically satisfied by small networks. Although they could be easily incorporated into our inductive definition, more than BNF style, they would obscure the relatively simple structure of our network specifications.

We only briefly explain what the second condition specifies: To avoid ambiguities in the formal semantics of Section 4, we need to enforce in the specification of a network $\mathcal{N}$ that no arc name refers to two different arcs. This in turn requires that we distinguish the arcs of the different copies of the same hole X. Thus, if we use $k \geqslant 2$ copies of X, we rename their arcs so that each copy has its own set of arcs. We write ${}^{1}X,\dots,{}^{k}X$ to refer to these k copies of X. For further details on the *unique arc-naming* condition, and full explanation of the two other conditions, the reader is referred to [19].

Example 7. We illustrate several of the notions introduced so far. We use one hole X, and 4 small networks: **F** ("fork"), **M** ("merge"), $\mathcal{A}$, and $\mathcal{B}$. These will be used again in later examples. We do not assign lower-bound and upper-bound capacities to the arcs of **F**, **M**, $\mathcal{A}$, and $\mathcal{B}$ – the arcs of holes are never assigned capacities – because they play no role before our typing theory is introduced. Graphic representations of **F**, **M**, and X are shown in Figure 1, and of $\mathcal{A}$ and $\mathcal{B}$ in Figure 2. A possible network specification $\mathcal{N}$ with two bound occurrences of X may read as follows:

$$\mathcal{N} = \textbf{let}\ X \in \{\mathcal{A},\mathcal{B}\}\ \textbf{in}\ \ \textbf{conn}(\ \textbf{F},\ \textbf{conn}(\ {}^{1}X,\ \textbf{conn}(\ {}^{2}X,\ \textbf{M},\theta_3),\theta_2),\theta_1)$$

where $\theta_1 = \{\langle c_2, {}^{1}e_1\rangle, \langle c_3, {}^{1}e_2\rangle\}$, $\theta_2 = \{\langle {}^{1}e_3, {}^{2}e_1\rangle, \langle {}^{1}e_4, {}^{2}e_2\rangle\}$, and $\theta_3 = \{\langle {}^{2}e_3, d_1\rangle, \langle {}^{2}e_4, d_2\rangle\}$. We wrote $\mathcal{N}$ above using some of the derived constructors introduced in Section 3.1. Note that:

- all the output arcs $\{c_2, c_3\}$ of **F** are connected to all the input arcs $\{{}^{1}e_1, {}^{1}e_2\}$ of ${}^{1}X$,
- all the output arcs $\{{}^{1}e_3, {}^{1}e_4\}$ of ${}^{1}X$ are connected to all the input arcs $\{{}^{2}e_1, {}^{2}e_2\}$ of ${}^{2}X$,
- all the output arcs $\{{}^{2}e_3, {}^{2}e_4\}$ of ${}^{2}X$ are connected to all the input arcs $\{d_1, d_2\}$ of **M**,

Hence, according to Section 3.1, we can write more simply:

$$\mathcal{N} = \textbf{let}\ X \in \{\mathcal{A},\mathcal{B}\}\ \textbf{in}\ \Big(\textbf{F} \oplus {}^{1}X \oplus {}^{2}X \oplus \textbf{M}\Big)$$

with now $\mathbf{in}(\mathcal{N}) = \{c_1\}$ and $\mathbf{out}(\mathcal{N}) = \{d_3\}$. The specification $\mathcal{N}$ says that $\mathcal{A}$ or $\mathcal{B}$ can be selected for insertion wherever hole X occurs. Though we do not define the reduction of **let-in**-bindings formally, $\mathcal{N}$ can be viewed as representing two different network configurations:

$$\mathcal{N}_1 = \mathbf{F} \oplus {}^1\mathcal{A} \oplus {}^2\mathcal{A} \oplus \mathbf{M} \quad \text{and} \quad \mathcal{N}_2 = \mathbf{F} \oplus {}^1\mathcal{B} \oplus {}^2\mathcal{B} \oplus \mathbf{M}$$

We can say nothing here about properties, such as safety, being satisfied or violated by these two configurations. The semantics of our **let-in** constructor later will be equivalent to requiring that both configurations be "safe" to use. By contrast, the constructor **try-in** mentioned in Remark 6 requires only $\mathcal{N}_1$ or $\mathcal{N}_2$, but not necessarily both, to be safe, and the constructor **mix-in** additionally requires:

$$\mathcal{N}_3 = \mathbf{F} \oplus {}^1\mathcal{A} \oplus {}^2\mathcal{B} \oplus \mathbf{M} \quad \text{and} \quad \mathcal{N}_4 = \mathbf{F} \oplus {}^1\mathcal{B} \oplus {}^2\mathcal{A} \oplus \mathbf{M}$$

to be safe. Safe substitution into holes according to **mix-in** implies safe substitution according to **let-in**, which in turn implies safe substitution according to **try-in**. □

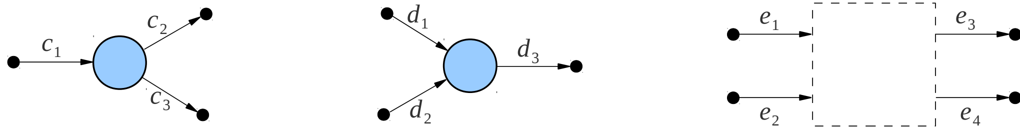

Figure 1: Small network **F** (on the left), small network **M** (in the middle), and hole X (on the right), in Example 7.

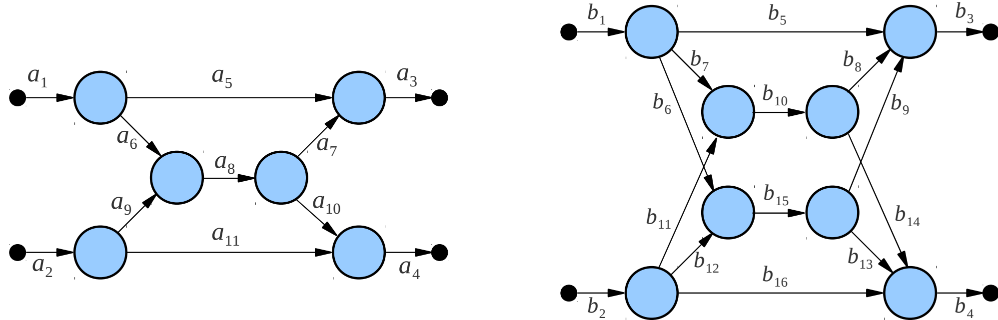

Figure 2: Small networks $\mathcal{A}$ (on the left) and $\mathcal{B}$ (on the right) in Example 7.

4 Formal Semantics of Flow Networks

The preceding section explained what we need to write to specify a network formally. Let $\mathcal{N}$ be such a network specification. By well-formedness, every small network $\mathcal{A}$ appearing in $\mathcal{N}$ has its own separate set of arc names, and every bound occurence ${}^iX$ of a hole X also has its own separate set of arc names, where $i \geqslant 1$ is a renaming index. (Renaming indeces are defined in Section 3.2.) With every small network $\mathcal{A}$, we associate two sets of functions, its *full semantics* $[\![\mathcal{A}]\!]$ and its *IO-semantics* $\langle\!\langle\mathcal{A}\rangle\!\rangle$. Let $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{A})$, $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{A})$, and $\mathbf{A}_{\#} = \#(\mathcal{A})$. The sets $[\![\mathcal{A}]\!]$ and $\langle\!\langle\mathcal{A}\rangle\!\rangle$ are defined thus:

$$[\![\mathcal{A}]\!] = \{ f : \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} \uplus \mathbf{A}_{\#} \to \mathbb{R}^+ \mid f \text{ is a feasible flow in } \mathcal{A} \}$$
$$\langle\!\langle\mathcal{A}\rangle\!\rangle = \{ f : \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} \to \mathbb{R}^+ \mid f \text{ can be extended to a feasible flow } f' \text{ in } \mathcal{A} \}$$

Let X be a hole, with $\mathbf{in}(X) = \mathbf{A}_{\text{in}}$ and $\mathbf{out}(X) = \mathbf{A}_{\text{out}}$. The *full semantics* $[\![X]\!]$ and the *IO-semantics* $\langle\!\langle X \rangle\!\rangle$ are the same set of functions:

$$[\![X]\!] \;=\; \langle\!\langle X \rangle\!\rangle \;\subseteq\; \{ f : \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} \to \mathbb{R}^+ \mid f \text{ is a bounded function} \}$$

This definition of $[\![X]\!] = \langle\!\langle X \rangle\!\rangle$ is ambiguous: In contrast to the uniquely defined full semantics and IO-semantics of a small network $\mathcal{A}$, there are infinitely many $[\![X]\!] = \langle\!\langle X \rangle\!\rangle$ for the same X, but exactly one (possibly $[\![X]\!] = \langle\!\langle X \rangle\!\rangle = \varnothing$) will satisfy the requirement in clause 4 below.

Starting from the full semantics of small networks and holes, we define by induction the full semantics $[\![\mathcal{N}]\!]$ of a network specification $\mathcal{N}$ in general. In a similar way, we can define the IO-semantics $\langle\!\langle \mathcal{N} \rangle\!\rangle$ of $\mathcal{N}$ by induction, starting from the IO-semantics of small networks and holes. For conciseness, we define $[\![\mathcal{N}]\!]$ separately first, and then define $\langle\!\langle \mathcal{N} \rangle\!\rangle$ from $[\![\mathcal{N}]\!]$. We need a few preliminary notions. Let $\mathcal{M}$ be a network specification. By our convention of listing all input arcs first, all output arcs second, and all internal arcs third, let:

$$\mathbf{in}(\mathcal{M}) = \{a_1,\ldots,a_k\}, \quad \mathbf{out}(\mathcal{M}) = \{a_{k+1},\ldots,a_{k+\ell}\}, \quad \text{and } \#(\mathcal{M}) = \{a_{k+\ell+1},\ldots,a_{k+\ell+m}\}.$$

If $f \in [\![\mathcal{M}]\!]$ with $f(a_1) = r_1,\ldots,f(a_{k+\ell+m}) = r_{k+\ell+m}$, we may represent f by the sequence $\langle r_1,\ldots,r_{k+\ell+m}\rangle$. We may therefore represent:

- $[f]_{\mathbf{in}(\mathcal{M})}$ by the sequence $\langle r_1,\ldots,r_k\rangle$,
- $[f]_{\mathbf{out}(\mathcal{M})}$ by the sequence $\langle r_{k+1},\ldots,r_{k+\ell}\rangle$, and
- $[f]_{\#(\mathcal{M})}$ by the sequence $\langle r_{k+\ell+1},\ldots,r_{k+\ell+m}\rangle$,

where $[f]_{\mathbf{in}(\mathcal{M})}$, $[f]_{\mathbf{out}(\mathcal{M})}$, and $[f]_{\#(\mathcal{M})}$, are the restrictions of f to the subsets $\mathbf{in}(\mathcal{M})$, $\mathbf{out}(\mathcal{M})$, and $\#(\mathcal{M})$, of its domain. Let $\mathcal{N}$ be another network specification and $g \in [\![\mathcal{N}]\!]$. We define $f \parallel g$ as follows:

$$(f \parallel g) = [f]_{\mathbf{in}(\mathcal{M})} \cdot [g]_{\mathbf{in}(\mathcal{N})} \cdot [f]_{\mathbf{out}(\mathcal{M})} \cdot [g]_{\mathbf{out}(\mathcal{N})} \cdot [f]_{\#(\mathcal{M})} \cdot [g]_{\#(\mathcal{N})}$$

where "$\cdot$" is sequence concatenation. The operation "$\parallel$" on flows is associative, but not commutative, just as the related constructor "$\parallel$" on network specifications. We define the full semantics $[\![\mathcal{M}]\!]$ for every subexpression $\mathcal{M}$ of $\mathcal{N}$, by induction on the structure of the specification $\mathcal{N}$:

1. If $\mathcal{M} = \mathcal{A}$, then $[\![\mathcal{M}]\!] = [\![\mathcal{A}]\!]$.
2. If $\mathcal{M} = {}^{i}X$, then $[\![\mathcal{M}]\!] = {}^{i}[\![X]\!]$.
3. If $\mathcal{M} = \big(\mathcal{P}_1 \parallel \mathcal{P}_2\big)$, then $[\![\mathcal{M}]\!] \;=\; \big\{ (f_1 \parallel f_2) \,\big|\, f_1 \in [\![\mathcal{P}_1]\!] \text{ and } f_2 \in [\![\mathcal{P}_2]\!] \big\}$.
4. If $\mathcal{M} = \big(\textsf{let } X = \mathcal{P} \textsf{ in } \mathcal{P}'\big)$, then $[\![\mathcal{M}]\!] = [\![\mathcal{P}']\!]$, provided two conditions:[3]
   (a) $\mathsf{dim}(X) \;\approx\; \mathsf{dim}(\mathcal{P})$,
   (b) $[\![X]\!] \;\approx\; \{ [g]_A \mid g \in [\![\mathcal{P}]\!] \}$ where $A = \mathbf{in}(\mathcal{P}) \cup \mathbf{out}(\mathcal{P})$.
5. If $\mathcal{M} = \textsf{bind}\,(\mathcal{P}, \langle a,b\rangle)$, then $[\![\mathcal{M}]\!] \;=\; \big\{ f \,\big|\, f \in [\![\mathcal{P}]\!] \text{ and } f(a) = f(b) \big\}$.

[3]"$\mathsf{dim}(X) \approx \mathsf{dim}(\mathcal{P})$" means the number of input arcs and their ordering (or input dimension) *and* the number of output arcs and their ordering (or output dimension) of X match those of $\mathcal{P}$, up to arc renaming (or dimension renaming). Similarly, "$[\![X]\!] \approx \{[g]_A | g \in [\![\mathcal{P}]\!]\}$" means for every $f : \mathbf{in}(X) \uplus \mathbf{out}(X) \to \mathbb{R}^+$, it holds that $f \in [\![X]\!]$ iff there is $g \in [\![\mathcal{P}]\!]$ such that $f \approx [g]_A$, where $[g]_A$ is the restriction of g to the subset A of its domain.

All of $\mathcal{N}$ is a special case of a subexpression of $\mathcal{N}$, so that a the semantics of $\mathcal{N}$ is simply $[\![\mathcal{N}]\!]$. Note, in clause 2, that all bound occurrences ${}^{i}X$ of the same hole X are assigned the same semantics $[\![X]\!]$, up to renaming of arc names. We can now define the IO-semantics of $\mathcal{N}$ as follows:

$$\langle\!\langle \mathcal{N} \rangle\!\rangle \;=\; \big\{ \, [f]_A \, \big| \, f \in [\![\mathcal{N}]\!] \, \big\}$$

where $A = \mathbf{in}(\mathcal{N}) \cup \mathbf{out}(\mathcal{N})$ and $[f]_A$ is the restriction of f to A.

Remark 8. For every small network $\mathcal{A}$ appearing in a network specification $\mathcal{N}$, the lower-bound and upper-bound functions, $L_{\mathcal{A}}$ and $U_{\mathcal{A}}$, are already defined. The lower-bound and upper-bound for all of $\mathcal{N}$, denoted $L_{\mathcal{N}}$ and $U_{\mathcal{N}}$, are then assembled from those for all the small networks. However, we do not need to explicitly define $L_{\mathcal{N}}$ and $U_{\mathcal{N}}$ at every step of the inductive definition of $\mathcal{N}$.

In clause 4, the lower-bound and upper-bound capacities on an input/output arc a of the hole X are determined by those on the corresponding arc, say a', in $\mathcal{P}$. Specifically, $L_X(a) = L_{\mathcal{P}}(a')$ and $U_X(a) = U_{\mathcal{P}}(a')$. In clause 5, the lower-bound and upper-bound are implicitly set. Specifically, consider output arc a and input arc b in $\mathcal{P}$, with $L_{\mathcal{P}}$ and $U_{\mathcal{P}}$ already defined on a and b. If $\mathcal{M} = \textbf{bind}\,(\mathcal{P}, \langle a,b\rangle)$, then:

$$\begin{aligned} L_{\mathcal{M}}(a) \;&=\; \max\,\{L_{\mathcal{P}}(a), L_{\mathcal{P}}(b)\} \\ U_{\mathcal{M}}(a) \;&=\; \min\,\{U_{\mathcal{P}}(a), U_{\mathcal{P}}(b)\} \end{aligned}$$

which are implied by the requirement that $f(a) = f(b)$. In $\mathcal{M}$, arc a is now internal and arc b is altogether omitted. On all the arcs other than a, $L_{\mathcal{M}}$ and $U_{\mathcal{M}}$ are identical to $L_{\mathcal{P}}$ and $U_{\mathcal{P}}$, respectively. □

Remark 9. We can define rewrite rules on network specifications in order to reduce each into an equivalent finite set of network specifications in *normal form*, a normal form being free of **try-in** bindings. We can do this so that the formal semantics of network specifications are an *invariant* of this rewriting. This establishes the *soundness* of the *operational semantics* (represented by the rewrite rules) of our DSL relative to the formal semantics defined above. We avoid formulating and presenting such rewriting rules in this report, for reasons alluded to in the Introduction and again in the last section. □

Flow Conservation, Capacity Constraints, Type Satisfaction (Continued). The fundamental concepts stated in relation to small networks $\mathcal{A}$ in Definitions 2, 3, and 4, are extended to arbitrary network specifications $\mathcal{N}$. These are stated as "properties" (not "definitions") because they apply to $[\![\mathcal{N}]\!]$ (not to $\mathcal{N}$), and $[\![\mathcal{N}]\!]$ is built up inductively from $\{[\![\mathcal{A}]\!] \mid \mathcal{A}$ occurs in $\mathcal{N}\}$.

Property 10 (Flow Conservation – Continued)**.** *The nodes of $\mathcal{N}$ are all the nodes in the small networks occurring in $\mathcal{N}$, because our DSL in Section 3 does not introduce new nodes beyond those in the small networks. Hence, $[\![\mathcal{N}]\!]$ satisfies* flow conservation *because, for every small network $\mathcal{A}$ in $\mathcal{N}$, every $f \in [\![\mathcal{A}]\!]$ satisfies flow conservation at every node, i.e., the equation in (1) in Definition 2.*

Property 11 (Capacity Constraints – Continued)**.** *The arcs introduced by our DSL, beyond the arcs in the small networks, are the input/output arcs of the holes. Lower-bound and upper-bound capacities on the latter arcs are set in order not to conflict with those already defined on the input/output arcs of small networks. Hence, $[\![\mathcal{N}]\!]$ satisfies* the capacity constraints *because, for every small network $\mathcal{A}$ in $\mathcal{N}$, every $f \in [\![\mathcal{A}]\!]$ satisfies the capacity constraints on every arc, i.e., the inequalities in (2) in Definition 3.*

However, stressing the obvious, even if $[\![\mathcal{A}]\!] \neq \varnothing$ for every small network $\mathcal{A}$ in $\mathcal{N}$, it may still be that $\mathcal{N}$ is unsafe to use, *i.e.*, it may still be that there is no feasible flow in $\mathcal{N}$ because $[\![\mathcal{N}]\!] = \varnothing$. We use the type system (Section 7) to reject unsafe network specifications $\mathcal{N}$.

Definition 12 (*Type Satisfaction – Continued*)**.** Let $\mathcal{N}$ be a network, with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$, $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$, and $\mathbf{A}_{\#} = \#(\mathcal{N})$. A typing T for $\mathcal{N}$, also denoted $(\mathcal{N} : T)$, is a function

$$T : \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \to \mathbb{R} \times \mathbb{R}$$

which may, or may not, be satisfied by $f \in \langle\!\langle \mathcal{N} \rangle\!\rangle$ or by $f \in [\![\mathcal{N}]\!]$. We say $f \in \langle\!\langle \mathcal{N} \rangle\!\rangle$ or $f \in [\![\mathcal{N}]\!]$ *satisfies* T iff, for every $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$ with $T(A) = [r, r']$, it is the case that:

(4) $$r \leqslant \quad \sum f(A \cap \mathbf{A}_{\text{in}}) - \sum f(A \cap \mathbf{A}_{\text{out}}) \quad \leqslant r'$$

The inequalities in (4) extend those in (3) in Definition 5 to network specifications in general. □

5 Typings Are Polytopes

Let $\mathcal{N}$ be a network specification, and let $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. Let T be a typing for $\mathcal{N}$ that assigns an interval $[r, r']$ to $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$. Let $|\mathbf{A}_{\text{in}}| + |\mathbf{A}_{\text{out}}| = m$, for some $m \geqslant 0$. As usual, there is a fixed ordering on the arcs in $\mathbf{A}_{\text{in}}$ and again on the arcs in $\mathbf{A}_{\text{out}}$. With no loss of generality, suppose:

$$A_1 = A \cap \mathbf{A}_{\text{in}} = \{a_1, \ldots, a_k\} \quad \text{and} \quad A_2 = A \cap \mathbf{A}_{\text{out}} = \{a_{k+1}, \ldots, a_\ell\},$$

where $\ell \leqslant m$. Instead of writing $T(A) = [r, r']$, we may write:

$$T(A): \quad a_1 + \cdots + a_k - a_{k+1} - \cdots - a_\ell \ : \ [r, r']$$

where the inserted polarities, + or −, indicate whether the arcs are input or output, respectively. A flow through the arcs $\{a_1, \ldots, a_k\}$ contributes a *positive* quantity, and through the arcs $\{a_{k+1}, \ldots, a_\ell\}$ a *negative* quantity, and these two quantities together should add up to a value within the interval $[r, r']$.

A typing T for $\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$ induces a *polytope* (or *bounded polyhedron*), which we call $\mathsf{Poly}(T)$, in the Euclidean hyperspace $\mathbb{R}^m$. We think of the m arcs in $\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$ as the m dimensions of the space $\mathbb{R}^m$. $\mathsf{Poly}(T)$ is the non-empty intersection of at most $2 \cdot (2^m - 1)$ halfspaces, because there are $(2^m - 1)$ non-empty subsets in $\mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$. The interval $[r, r']$, which T assigns to such a subset $A = \{a_1, \ldots, a_\ell\}$ as above, induces two linear inequalities in the variables $\{a_1, \ldots, a_\ell\}$, denoted $T_{\geqslant}(A)$ and $T_{\leqslant}(A)$:

(5) $$T_{\geqslant}(A): \quad a_1 + \cdots + a_k - a_{k+1} - \cdots - a_\ell \geqslant r \qquad \text{and} \qquad T_{\leqslant}(A): \quad a_1 + \cdots + a_k - a_{k+1} - \cdots - a_\ell \leqslant r'$$

and, therefore, two halfspaces $\mathsf{Half}(T_{\geqslant}(A))$ and $\mathsf{Half}(T_{\leqslant}(A))$:

(6) $$\mathsf{Half}(T_{\geqslant}(A)) = \{ \boldsymbol{r} \in \mathbb{R}^m \mid \boldsymbol{r} \text{ satisfies } T_{\geqslant}(A) \} \qquad \text{and} \qquad \mathsf{Half}(T_{\leqslant}(A)) = \{ \boldsymbol{r} \in \mathbb{R}^m \mid \boldsymbol{r} \text{ satisfies } T_{\leqslant}(A) \}$$

We can therefore define $\mathsf{Poly}(T)$ formally as follows:

$$\boxed{\mathsf{Poly}(T) = \bigcap \big\{ \mathsf{Half}(T_{\geqslant}(A)) \cap \mathsf{Half}(T_{\leqslant}(A)) \ \big| \ \varnothing \neq A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \big\}}$$

Generally, many of the inequalities induced by the typing T will be redundant, and the induced $\mathsf{Poly}(T)$ will be defined by far fewer than $2 \cdot (2^m - 1)$ halfspaces.

5.1 Uniqueness and Redundancy in Typings

We can view a network typing T as a syntactic expression, with its semantics $\mathsf{Poly}(T)$ being a polytope in Euclidean hyperspace. As in other situations connecting syntax and semantics, there are generally distinct typings T and T' such that $\mathsf{Poly}(T) = \mathsf{Poly}(T')$. This is an obvious consequence of the fact that the same polytope can be defined by many different equivalent sets of linear inequalities, which is the source of some complications when we combine two typings to produce a new one.

To achieve uniqueness of typings, as well as some efficiency of manipulating them, we may try an approach that eliminates redundant inequalities in the collection:

$$(7) \quad \{\, T_{\geqslant}(A) \mid \varnothing \neq A \in \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})\,\} \;\cup\; \{\, T_{\leqslant}(A) \mid \varnothing \neq A \in \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})\,\}$$

where $T_{\geqslant}(A)$ and $T_{\leqslant}(A)$ are as in (5) above. There are standard procedures which determine whether a finite set of inequalities are linearly independent and, if they are not, select an equivalent subset of linearly independent inequalities. Some of these issues are taken up in the full report [19].

If $\mathcal{N}_1 : T_1$ and $\mathcal{N}_2 : T_2$ are typings for networks $\mathcal{N}_1$ and $\mathcal{N}_2$ with matching input and output dimensions, we write $T_1 \equiv T_2$ whenever $\mathsf{Poly}(T_1) \approx \mathsf{Poly}(T_2)$, in which case we say that T_1 and T_2 are *equivalent*.[4] If $\mathcal{N}_1 = \mathcal{N}_2$, then $T_1 \equiv T_2$ whenever $\mathsf{Poly}(T_1) = \mathsf{Poly}(T_2)$.

Definition 13 (*Tight Typings*)**.** Let $\mathcal{N}$ be a network specification, with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$, and $T : \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \to \mathbb{R} \times \mathbb{R}$ a typing for $\mathcal{N}$. T is a *tight* typing if for every typing T' such that $T \equiv T'$ and for every $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, the interval $T(A)$ is contained in the interval $T'(A)$, *i.e.*, $T(A) \subseteq T'(A)$. □

Proposition 14 (Every Typing Is Equivalent to a Tight Typing)**.** *There is an algorithm* $\mathsf{Tight}()$ *which, given a typing* $(\mathcal{N} : T)$ *as input, always terminates and returns an equivalent tight typing* $(\mathcal{N} : \mathsf{Tight}(T))$.

5.2 Valid Typings and Principal Typings

Let $\mathcal{N}$ be a network, $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. A typing $\mathcal{N} : T$ is *valid* iff it is sound:

(soundness) Every $f_0 : \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \to \mathbb{R}^+$ satisfying T can be extended to a feasible flow $f \in [\![\mathcal{N}]\!]$.

We say the typing $\mathcal{N} : T$ for $\mathcal{N}$ is a *principal typing* if it is both sound *and* complete:

(completeness) Every feasible flow $f \in [\![\mathcal{N}]\!]$ satisfies T.

More succintly, using the IO-semantics $\langle\!\langle \mathcal{N} \rangle\!\rangle$ instead of the full semantics $[\![\mathcal{N}]\!]$, the typing $\mathcal{N} : T$ is *valid* iff $\mathsf{Poly}(T) \subseteq \langle\!\langle \mathcal{N} \rangle\!\rangle$, and it is *principal* iff $\mathsf{Poly}(T) = \langle\!\langle \mathcal{N} \rangle\!\rangle$.

A useful notion in type theories is *subtyping*. If T_1 is a *subtype* of T_2, in symbols $T_1 <: T_2$, this means that any object of type T_1 can be safely used in a context where an object of type T_2 is expected:

(subtyping) $\quad T_1 <: T_2 \quad$ iff $\quad \mathsf{Poly}(T_2) \subseteq \mathsf{Poly}(T_1)$.

Our subtyping relation is contravariant w.r.t. the subset relation, *i.e.*, the supertype T_2 is more restrictive as a set of flows than the subtype T_1.

Proposition 15 (Principal Typings Are Subtypes of Valid Typings)**.** *If* $(\mathcal{N} : T_1)$ *is a principal typing, and* $(\mathcal{N} : T_2)$ *a valid typing for the same* $\mathcal{N}$*, then* $T_1 <: T_2$.

Any two principal typings T_1 and T_2 of the same network are not necessarily identical, but they always denote the same polytope, as formally stated in the next proposition.

Proposition 16 (Principal Typings Are Equivalent)**.** *If* $(\mathcal{N} : T_1)$ *and* $(\mathcal{N} : T_2)$ *are two principal typings for the same network specification* $\mathcal{N}$*, then* $T_1 \equiv T_2$*. Moreover, if* T_1 *and* T_2 *are tight, then* $T_1 = T_2$.

[4] "$\mathsf{Poly}(T_1) \approx \mathsf{Poly}(T_2)$" means that $\mathsf{Poly}(T_1)$ and $\mathsf{Poly}(T_2)$ are the same up to renaming their dimensions, *i.e.*, up to renaming the input and output arcs in $\mathcal{N}_1$ and $\mathcal{N}_2$.

6 Inferring Typings for Small Networks

Theorem 17 (Existence of Principal Typings)**.** *Let $\mathcal{A}$ be a small network. We can effectively compute a principal and uniformly tight typing T for $\mathcal{A}$.*

Example 18. Consider again the two small networks $\mathcal{A}$ and $\mathcal{B}$ from Example 7. We assign capacities to their arcs and compute their respective principal typings. The sets of arcs in $\mathcal{A}$ and $\mathcal{B}$ are, respectively: $\mathbf{A} = \{a_1,\ldots,a_{11}\}$ and $\mathbf{B} = \{b_1,\ldots,b_{16}\}$. All the lower-bounds and most of the upper-bounds are trivial, *i.e.*, they do not restrict flow. Specifically, the lower-bound capacity on every arc is 0, and the upper-bound capacity on every arc is a "very large number", unless indicated otherwise in Figure 3 by the numbers in rectangular boxes, namely:

$$
\begin{array}{lllll}
U(a_5) = 5, & U(a_8) = 10, & U(a_{11}) = 15, & & \text{non-trivial upper-bounds in } \mathcal{A}, \\
U(b_5) = 3, & U(b_6) = 2, & U(b_9) = 2, & U(b_{10}) = 10, & \text{non-trivial upper-bounds in } \mathcal{B}, \\
U(b_{11}) = 8, & U(b_{13}) = 8, & U(b_{15}) = 10, & U(b_{16}) = 7, & \text{non-trivial upper-bounds in } \mathcal{B}.
\end{array}
$$

We compute the principal typings $T_{\mathcal{A}}$ of $\mathcal{A}$ and $T_{\mathcal{B}}$ of $\mathcal{B}$, by assigning a bounded interval to every subset of $\{a_1,a_2,a_3,a_4\}$ and $\{b_1,b_2,b_3,b_4\}$, respectively. This is a total of 15 intervals for each, ignoring the empty set to which we assign the empty interval $\varnothing$. We use the construction in the proof (omitted in this paper, included in the full report [19]) of Theorem 17 to compute $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$.

$T_{\mathcal{A}}$ assignments:

$$
\begin{array}{llll}
\boxed{a_1 : [0,15]} & \boxed{a_2 : [0,25]} & \boxed{-a_3 : [-15,0]} & \boxed{-a_4 : [-25,0]} \\
\boxed{a_1 + a_2 : [0,30]} & \underline{a_1 - a_3 : [-10,10]} & a_1 - a_4 : [-25,15] & \\
a_2 - a_3 : [-15,25] & \underline{a_2 - a_4 : [-10,10]} & \boxed{-a_3 - a_4 : [-30,0]} & \\
a_1 + a_2 - a_3 : [0,25] & a_1 + a_2 - a_4 : [0,15] & a_1 - a_3 - a_4 : [-25,0] & a_2 - a_3 - a_4 : [-15,0] \\
a_1 + a_2 - a_3 - a_4 : [0,0] & & &
\end{array}
$$

$T_{\mathcal{B}}$ assignments:

$$
\begin{array}{llll}
\boxed{b_1 : [0,15]} & \boxed{b_2 : [0,25]} & \boxed{-b_3 : [-15,0]} & \boxed{-b_4 : [-25,0]} \\
\boxed{b_1 + b_2 : [0,30]} & \underline{b_1 - b_3 : [-10,12]} & b_1 - b_4 : [-25,15] & \\
b_2 - b_3 : [-15,25] & \underline{b_2 - b_4 : [-12,10]} & \boxed{-b_3 - b_4 : [-30,0]} & \\
b_1 + b_2 - b_3 : [0,25] & b_1 + b_2 - b_4 : [0,15] & b_1 - b_3 - b_4 : [-25,0] & b_2 - b_3 - b_4 : [-15,0] \\
b_1 + b_2 - b_3 - b_4 : [0,0] & & &
\end{array}
$$

The types in rectangular boxes are those of $[T_{\mathcal{A}}]_{\text{in}}$ and $[T_{\mathcal{B}}]_{\text{in}}$ which are equivalent, and those of $[T_{\mathcal{A}}]_{\text{out}}$ and $[T_{\mathcal{B}}]_{\text{out}}$ which are also equivalent. Thus, $[T_{\mathcal{A}}]_{\text{in}} \equiv [T_{\mathcal{B}}]_{\text{in}}$ and $[T_{\mathcal{A}}]_{\text{out}} \equiv [T_{\mathcal{B}}]_{\text{out}}$. Nevertheless, $T_{\mathcal{A}} \not\equiv T_{\mathcal{B}}$, the difference being in the (underlined) types assigned to some subsets mixing input and output arcs:

- $[-10,10]$ assigned by $T_{\mathcal{A}}$ to $\{a_1,a_3\}$ $\neq$ $[-10,12]$ assigned by $T_{\mathcal{B}}$ to the corresponding $\{b_1,b_3\}$,
- $[-10,10]$ assigned by $T_{\mathcal{A}}$ to $\{a_2,a_4\}$ $\neq$ $[-12,10]$ assigned by $T_{\mathcal{B}}$ to the corresponding $\{b_2,b_4\}$.

In this example, $T_{\mathcal{B}} <: T_{\mathcal{A}}$ because $\mathsf{Poly}(T_{\mathcal{A}}) \subseteq \mathsf{Poly}(T_{\mathcal{B}})$. The converse does not hold. As a result, there are feasible flows in $\mathcal{B}$ which are not feasible flows in $\mathcal{A}$. □

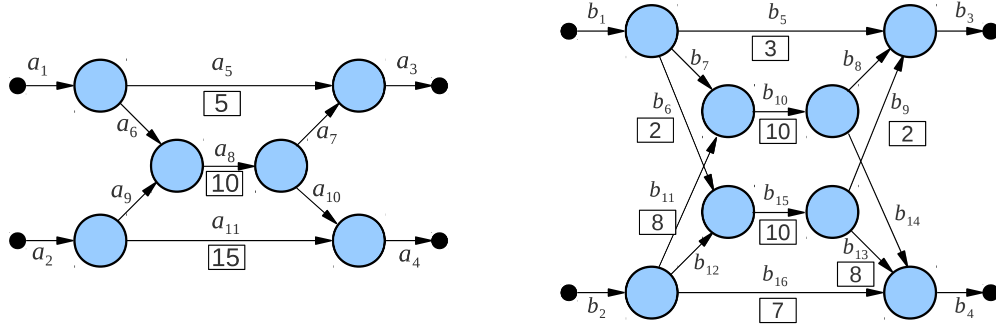

Figure 3: An assignment of arc capacities for small networks $\mathcal{A}$ (on the left) and $\mathcal{B}$ (on the right) in Example 18.

7 A Typing System

We set up a formal system for assigning typings to network specifications. The process of inferring typings, based on this system, is deferred to Section 8. We need several preliminary definitions.

7.1 Operations on Typings

Let $(\mathcal{N}_1 : T_1)$ and $(\mathcal{N}_2 : T_2)$ be two typings for two networks $\mathcal{N}_1$ and $\mathcal{N}_2$. The four arc sets: $\mathbf{in}(\mathcal{N}_1)$, $\mathbf{out}(\mathcal{N}_1)$, $\mathbf{in}(\mathcal{N}_2)$, and $\mathbf{out}(\mathcal{N}_2)$, are pairwise disjoint. By our inductive definition in Section 3, $\mathbf{in}(\mathcal{N}_1) \cup \mathbf{in}(\mathcal{N}_2)$ is the set of input arcs, and $\mathbf{out}(\mathcal{N}_1) \cup \mathbf{out}(\mathcal{N}_2)$ the set of output arcs, for the network specification $\big(\mathcal{N}_1 \,\|\, \mathcal{N}_2\big)$. We define the typing $\big(T_1 \,\|\, T_2\big)$ for the specification $\big(\mathcal{N}_1 \,\|\, \mathcal{N}_2\big)$ as follows:

$$\big(T_1 \,\|\, T_2\big)(A) = \begin{cases} T_1(A) & \text{if } A \subseteq \mathbf{in}(\mathcal{N}_1) \cup \mathbf{out}(\mathcal{N}_1), \\[1em] T_2(A) & \text{if } A \subseteq \mathbf{in}(\mathcal{N}_2) \cup \mathbf{out}(\mathcal{N}_2), \\[1em] T_1(A_1) \oplus T_2(A_2) & \text{if } A = A_1 \cup A_2 \text{ where} \\ & A_1 \subseteq \mathbf{in}(\mathcal{N}_1) \cup \mathbf{out}(\mathcal{N}_1) \text{ and } A_2 \subseteq \mathbf{in}(\mathcal{N}_2) \cup \mathbf{out}(\mathcal{N}_2). \end{cases}$$

where the operation "$\oplus$" on intervals is defined as follows: $[r_1, r_2] \oplus [r'_1, r'_2] = [r_1 + r'_1, r_2 + r'_2]$.

Lemma 19. *If $(\mathcal{N}_1 : T_1)$ and $(\mathcal{N}_2 : T_2)$ are principal typings, respectively valid typings, then so is the typing $\big((\mathcal{N}_1 \,\|\, \mathcal{N}_2) : (T_1 \,\|\, T_2)\big)$ principal, respectively valid.*

Let $(\mathcal{N} : T)$ be a typing with $\langle a, b\rangle \in \mathbf{out}(\mathcal{N}) \times \mathbf{in}(\mathcal{N})$, with $\mathsf{dim}_{\text{in}}(\mathcal{N}) = \langle a_1, \ldots, a_\ell\rangle$ and $\mathsf{dim}_{\text{out}}(\mathcal{N}) = \langle a_{\ell+1}, \ldots, a_m\rangle$, so that $b = a_i$ and $a = a_j$ for some $1 \leqslant i \leqslant \ell$ and $\ell + 1 \leqslant j \leqslant m$. In the full report [19] we explain how to define a typing we denote $\mathsf{bind}(T, \langle a, b\rangle)$ from the given typing T for the network specification $\textsf{\textbf{bind}}\ (\mathcal{N}, \langle a, b\rangle)$ satisfying the equation: $\mathsf{Poly}(\mathsf{bind}(T, \langle a, b\rangle)) \ = \ \mathsf{Poly}(T) \cap \mathsf{Poly}(a = b)$ where

$$\mathsf{Poly}(a = b) \ = \ \{\langle r_1, \ldots, r_m\rangle \in \mathbb{R}^m \mid r_i = r_j\} \quad \text{where } b = a_i \text{ and } a = a_j \text{ with } 1 \leqslant i \leqslant \ell < j \leqslant m.$$

Lemma 20. *If $(\mathcal{N} : T)$ is a principal (respectively, valid) typing and $\langle a, b\rangle \in \boldsymbol{in}(\mathcal{N}) \times \boldsymbol{out}(\mathcal{N})$, then $\big(\textsf{\textbf{bind}}\ (\mathcal{N}, \langle a, b\rangle) : \mathsf{bind}(T, \langle a, b\rangle)\big)$ is a principal (respectively, valid) typing.*

7.2 Typing Rules

The system is in Figure 4, where we follow standard conventions in formulating the rules. We call Γ a *typing environment*, which is a finite set of *typing assumptions* for holes, each of the form $(X:T)$. If $(X:T)$ is a typing assumption, with $\mathbf{in}(X)=\mathbf{A}_{\text{in}}$ and $\mathbf{out}(X)=\mathbf{A}_{\text{out}}$, then $T:\mathscr{P}(\mathbf{A}_{\text{in}}\cup\mathbf{A}_{\text{out}})\to\mathbb{R}\times\mathbb{R}$.

If a typing T is derived for a network specification $\mathcal{N}$ according to the rules in Figure 4, it will be the result of deriving an *assertion* (or *judgment*) of the form "$\Gamma\vdash\mathcal{N}:T$". If $\mathcal{N}$ is closed, then this final typing judgment will be of the form "$\vdash\mathcal{N}:T$" where all typing assumptions have been discharged.

HOLE	$\dfrac{(X:T)\in\Gamma}{\Gamma\vdash {}^iX:{}^iT}$	$i\geqslant 1$ is the smallest available renaming index
SMALL	$\dfrac{}{\Gamma\vdash\mathcal{A}:T}$	T is a typing for small network $\mathcal{A}$
PAR	$\dfrac{\Gamma\vdash\mathcal{N}_1:T_1 \qquad \Gamma\vdash\mathcal{N}_2:T_2}{\Gamma\vdash(\mathcal{N}_1 \,\Vert\, \mathcal{N}_2):(T_1\parallel T_2)}$	
BIND	$\dfrac{\Gamma\vdash\mathcal{N}:T}{\Gamma\vdash \mathsf{bind}\,(\mathcal{N},\langle a,b\rangle):\mathsf{bind}(T,\langle a,b\rangle)}$	$\langle a,b\rangle\in\mathbf{out}(\mathcal{N})\times\mathbf{in}(\mathcal{N})$
LET	$\dfrac{\Gamma\vdash\mathcal{M}:T_1 \qquad \Gamma\cup\{(X:T_2)\}\vdash\mathcal{N}:T}{\Gamma\vdash(\ \mathsf{let}\ X=\mathcal{M}\ \mathsf{in}\ \mathcal{N}\ ):T}$	$T_1\approx T_2$

Figure 4: Typing Rules for Flow Networks.
The operations $(T_1\parallel T_2)$ and $\mathsf{bind}(T,\langle a,b\rangle)$ are defined in Section 7.1. A derivation according to the rules is stopped from the moment a judgment $\Gamma\vdash\mathcal{N}:T$ is reached such that $\mathsf{Poly}(T)=\varnothing$, at which point $\mathcal{N}$ is rejected as "unsafe".

Theorem 21 (Existence of Principal Typings)**.** *Let $\mathcal{N}$ be a closed network specification and T a typing for $\mathcal{N}$ derived according to the rules in Figure 4, i.e., the judgment "$\vdash\mathcal{N}:T$" is derivable according to the rules. If the typing of every small network $\mathcal{A}$ in $\mathcal{N}$ is principal (resp., valid) for $\mathcal{A}$, then T is a principal (resp., valid) typing for $\mathcal{N}$.*

8 Inferring Typings for Flow Networks in General

The main difficulty in typing inference is in relation to **let**-bindings. Consider a specification $\mathcal{N}$ of the form $(\mathsf{let}\ X=\mathcal{M}\ \mathsf{in}\ \mathcal{P})$. Let $\mathbf{A}_{\text{in}}=\mathbf{in}(X)$ and $\mathbf{A}_{\text{out}}=\mathbf{out}(X)$. Suppose X occurs $n\geqslant 1$ times in $\mathcal{P}$, so that its input/output arcs are renamed in each of the n occurrences according to: ${}^1(\mathbf{A}_{\text{in}}\cup\mathbf{A}_{\text{out}})$, ... , ${}^n(\mathbf{A}_{\text{in}}\cup\mathbf{A}_{\text{out}})$. A typing for X and for its occurrences ${}^iX$ in $\mathcal{P}$ can be given *concretely* or *symbolically*. If concretely, then these typings are functions of the form:

$$T_X:\mathscr{P}(\mathbf{A}_{\text{in}}\cup\mathbf{A}_{\text{out}})\to\mathbb{R}\times\mathbb{R} \quad \text{and} \quad {}^iT_X:\mathscr{P}({}^i\mathbf{A}_{\text{in}}\cup{}^i\mathbf{A}_{\text{out}})\to\mathbb{R}\times\mathbb{R}$$

for every $1\leqslant i\leqslant n$. According to the typing rule HOLE in Figure 4, a valid typing for $\mathcal{N}$ requires that: $T_X\approx {}^1T_X\approx\cdots\approx {}^nT_X$. If symbolically, then for every $B\subseteq\mathbf{A}_{\text{in}}\cup\mathbf{A}_{\text{out}}$, the interval $T_X(B)$ is written as $[x_B,y_B]$ where the two ends x_B and y_B are yet to be determined, and similarly for ${}^iT_X(B)$ and every $B\subseteq{}^i\mathbf{A}_{\text{in}}\cup{}^i\mathbf{A}_{\text{out}}$. We can infer a typing for $\mathcal{N}$ in one of two ways, which produce the same end result but whose organizations are very different:

(sequential) First infer a principal typing $T_{\mathcal{M}}$ for $\mathcal{M}$, then use k copies ${}^1T_{\mathcal{M}},\ldots,{}^nT_{\mathcal{M}}$ to infer a principal typing $T_{\mathcal{P}}$ for $\mathcal{P}$, which is also a principal typing $T_{\mathcal{N}}$ for $\mathcal{N}$.

(parallel) Infer principal typings $T_{\mathcal{M}}$ for $\mathcal{M}$ and $T_{\mathcal{P}}$ for $\mathcal{P}$, separately. $T_{\mathcal{P}}$ is parametrized by the typings ${}^{i}T_X$ written symbolically. A typing for $\mathcal{N}$ is obtained by setting lower-end and upper-end parameters in ${}^{i}T_X$ to corresponding lower-end and upper-end values in $T_{\mathcal{M}}$.

Both approaches are *modular*, in that both are syntax-directed according to the inductive definition of $\mathcal{N}$. However, the parallel approach has the advantage of being independent of the order in which the inference proceeds (*i.e.*, it does not matter whether $T_{\mathcal{M}}$ is inferred before or after, or simultaneously with, $T_{\mathcal{P}}$). We therefore qualify the parallel approach as being additionally *fully compositional*, in contrast to the sequential approach which is not. Moreover, the latter requires that the whole specification $\mathcal{N}$ be known before typing inference can start, justifying the additional qualification of being a *whole-specification* analysis. The sequential approach is simpler to define and is presented in full in [19]. We delay the examination of the parallel/fully-compositional approach to a follow-up report.

9 Semantics of Flow Networks Relative to Objective Functions

Let $\mathcal{N}$ be a network, with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$, $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$, and $\mathbf{A}_{\#} = \mathbf{\#}(\mathcal{N})$. We write $\mathbf{A}_{\text{out},\#}$ to denote $\mathbf{A}_{\text{out}} \uplus \mathbf{A}_{\#}$, the set of all arcs in $\mathcal{N}$ excluding the input arcs. An *objective function* selects a subset of feasible flows that minimize (or maximize) some quantity. We list two possible objective functions, among several others, commonly considered in "traffic engineering" (see [3] for example).

Minimize Hop Routing **(HR)** A minimum hop route is a route with minimal number of links.
Given a feasible flow $f \in [\![\mathcal{N}]\!]$, we define the quantity $\text{HR}(f) = \sum_{a \in \mathbf{A}_{\text{out},\#}} f(a)$. Given two feasible flows $f_1, f_2 \in [\![\mathcal{N}]\!]$, we write $f_1 <^{\text{HR}} f_2$ iff two conditions:

- $[f_1]_{\mathbf{A}_{\text{in}}} = [f_2]_{\mathbf{A}_{\text{in}}}$, and
- $\text{HR}(f_1) < \text{HR}(f_2)$.

Note that we compare f_1 and f_2 using $<^{\text{HR}}$ only if they assign the same values to the input arcs, which implies in particular that f_1 and f_2 carry equal flows across $\mathcal{N}$. It can be shown that $\text{HR}(f_1) < \text{HR}(f_2)$ holds iff f_1 is non-zero on fewer arcs in $\mathbf{A}_{\text{out},\#}$ than f_2, *i.e.*,

$$|\{ a \in \mathbf{A}_{\text{out},\#} \mid f_1(a) \neq 0 \}| \;<\; |\{ a \in \mathbf{A}_{\text{out},\#} \mid f_2(a) \neq 0 \}|$$

We write $f_1 \leqslant^{\text{HR}} f_2$ to mean $f_1 <^{\text{HR}} f_2$ or $\text{HR}(f_1) = \text{HR}(f_2)$.

Minimize Arc Utilization **(AU)** The utilization of an arc a is defined as $u(a) = f(a)/U(a)$.
Given a feasible flow $f \in [\![\mathcal{N}]\!]$, we define the quantity $\text{AU}(f) = \sum_{a \in \mathbf{A}_{\text{out},\#}} u(a)$. Given two feasible flows $f_1, f_2 \in [\![\mathcal{N}]\!]$, we write $f_1 <^{\text{AU}} f_2$ iff two conditions:

- $[f_1]_{\mathbf{A}_{\text{in}}} = [f_2]_{\mathbf{A}_{\text{in}}}$, and
- $\text{AU}(f_1) < \text{AU}(f_2)$.

It can be shown that $\text{AU}(f_1) < \text{AU}(f_2)$ holds iff:

$$\sum\{ 1/U(a) \mid a \in \mathbf{A}_{\text{out},\#} \text{ and } f_1(a) \neq 0 \} \;<\; \sum\{ 1/U(a) \mid a \in \mathbf{A}_{\text{out},\#} \text{ and } f_2(a) \neq 0 \}$$

Minimizing arc utilization corresponds to computing "shortest paths" from inputs to outputs using $1/U(a)$ as the metric on every arc in $\mathbf{A}_{\text{out},\#}$. We write $f_1 \leqslant^{\text{AU}} f_2$ to mean $f_1 <^{\text{AU}} f_2$ or $\text{AU}(f_1) = \text{AU}(f_2)$.

For the rest of this section, consider a fixed objective $\alpha \in \{\text{HR}, \text{AU}, \ldots\}$. We relativize the formal semantics of flow networks as presented in Section 4. To be correct, our relativized semantics requires that the objective α be an "additive aggregate function".

Definition 22 (*Additive Aggregate Functions*)**.** Let $\mathcal{N}$ be a network and consider its set $[\![\mathcal{N}]\!]$ of feasible flows. A function $\alpha : [\![\mathcal{N}]\!] \to \mathbb{R}^+$ is an *additive aggregate* if $\alpha(f)$ is of the form $\sum_{a \in \mathbf{A}_{\text{out},\#}} \theta(f,a)$ for some function $\theta : [\![\mathcal{N}]\!] \times \mathbf{A}_{\text{out},\#} \to \mathbb{R}^+$. □

The particular objective functions HR and AU considered above are additive aggregate. For HR, the corresponding function θ is the simplest and defined by $\theta(f,a) = f(a)$. And for AU, the corresponding function θ is defined by $\theta(f,a) = f(a)/U(a)$. All the objective functions considered in [3] are additive aggregate.

The *full semantics of a flow network $\mathcal{N}$ relative to objective α*, denoted $[\![\mathcal{N} \,|\, \alpha]\!]$, will be a set of triples each of the form $\langle f, B, r\rangle$ where:

- $f \in [\![\mathcal{N}]\!]$, *i.e.*, f is a feasible flow in $\mathcal{N}$,
- $B \subseteq \mathbf{in}(\mathcal{N}) \cup \mathbf{out}(\mathcal{N})$,
- $r = \alpha(f)$,

such that, for every feasible flow $g \in [\![\mathcal{N}]\!]$, if $[f]_B = [g]_B$ then $\alpha(g) \geqslant r$. The information provided by the parameters B and r allows us to determine $[\![\mathcal{N} \,|\, \alpha]\!]$ compositionally, *i.e.*, in clause 5 in the definition of $[\![\mathcal{N} \,|\, \alpha]\!]$ below: We can define the semantics of a network $\mathcal{M}$ relative to α from the semantics of its *immediate* constituent parts relative to α. Informally, if $\langle f, B, r\rangle \in [\![\mathcal{N} \,|\, \alpha]\!]$, then among all feasible flows that agree on B, flow f minimizes $\alpha(f)$. We include the parameter $r = \alpha(f)$ in the triple to avoid re-computing α from scratch at every step of the induction, by having to sum over *all* the arcs of $\mathcal{N}$. Based on the preceding, starting with small networks $\mathcal{A}$, we define the full semantics of $\mathcal{A}$ relative to the objective α as follows:

$$[\![\mathcal{A} \,|\, \alpha]\!] = \Big\{ \langle f, B, r\rangle \;\Big|\; f \in [\![\mathcal{A}]\!],\ B \subseteq \mathbf{in}(\mathcal{A}) \cup \mathbf{out}(\mathcal{A}),\ r = \alpha(f), \text{ and for every } g \in [\![\mathcal{A}]\!], \text{ if } [f]_B = [g]_B \text{ then } \alpha(f) \leqslant \alpha(g) \Big\}$$

The IO-semantics $\langle\!\langle \mathcal{A} \,|\, \alpha \rangle\!\rangle$ of the small network $\mathcal{A}$ relative to the objective α is:

$$\langle\!\langle \mathcal{A} \,|\, \alpha \rangle\!\rangle = \Big\{ \langle [f]_A, B, r\rangle \;\Big|\; \langle f, B, r\rangle \in [\![\mathcal{A} \,|\, \alpha]\!] \Big\}$$

where $A = \mathbf{in}(\mathcal{A}) \cup \mathbf{out}(\mathcal{A})$. As in Section 4, the full semantics $[\![X \,|\, \alpha]\!]$ and the IO-semantics $\langle\!\langle X \,|\, \alpha \rangle\!\rangle$ of a hole X relative to the objective α are the same. Let $\mathbf{A}_{\text{in}} = \mathbf{in}(X)$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(X)$, so that:

$$[\![X \,|\, \alpha]\!] = \langle\!\langle X \,|\, \alpha \rangle\!\rangle \subseteq \Big\{ \langle f, B, s\rangle \;\Big|\; f : \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \to \mathbb{R}^+,\ B \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}},\ s \in \mathbb{R}^+, \text{ and } f \text{ is bounded} \Big\}$$

Again, as in Section 4, $[\![X \,|\, \alpha]\!] = \langle\!\langle X \,|\, \alpha \rangle\!\rangle$ is not uniquely defined. Whether this assigned semantics of X will work depends on whether the condition in clause 4 below is satisfied.

We define $[\![\mathcal{M} \,|\, \alpha]\!]$ for every subexpression $\mathcal{M}$ of $\mathcal{N}$, by induction on the structure of the specification $\mathcal{N}$. The five clauses here are identical to those in Section 4, except for the α-relativization. The only non-trivial clause is the 5th and last; Proposition 23 establishes the correctness of this definition:

1. If $\mathcal{M} = \mathcal{A}$, then $[\![\mathcal{M} \,|\, \alpha]\!] = [\![\mathcal{A} \,|\, \alpha]\!]$.
2. If $\mathcal{M} = {}^{i}X$, then $[\![\mathcal{M} \,|\, \alpha]\!] = {}^{i}[\![X \,|\, \alpha]\!]$.
3. If $\mathcal{M} = (\mathcal{P}_1 \parallel \mathcal{P}_2)$, then

$$[\![\mathcal{M} \,|\, \alpha]\!] = \Big\{ \langle f_1 \parallel f_2, B_1 \cup B_2, r_1 + r_2\rangle \;\Big|\; \langle f_1, B_1, r_1\rangle \in [\![\mathcal{P}_1 \,|\, \alpha]\!] \text{ and } \langle f_2, B_2, r_2\rangle \in [\![\mathcal{P}_2 \,|\, \alpha]\!] \Big\}$$

4. If $\mathcal{M} = \big(\textsf{let}\ X = \mathcal{P}\ \textsf{in}\ \mathcal{P}'\big)$, then $[\![\mathcal{M}\,|\,\alpha]\!] = [\![\mathcal{P}'\,|\,\alpha]\!]$, provided two conditions:[5]
   (a) $\mathsf{dim}(X) \approx \mathsf{dim}(\mathcal{P})$,
   (b) $[\![X\,|\,\alpha]\!] \approx \big\{ \langle [g]_A, C, r\rangle \,\big|\, \langle g, C, r\rangle \in [\![\mathcal{P}\,|\,\alpha]\!] \big\}$ where $A = \mathbf{in}(\mathcal{P}) \cup \mathbf{out}(\mathcal{P})$.
5. If $\mathcal{M} = \textsf{bind}\,(\mathcal{P}, \langle a,b\rangle)$, then

$$[\![\mathcal{M}\,|\,\alpha]\!] = \Big\{ \langle f,B,r\rangle \,\Big|\, \begin{array}{l} \langle f, B\cup\{a,b\}, r\rangle \in [\![\mathcal{P}\,|\,\alpha]\!],\ f(a) = f(b), \\ \text{and for every } \langle g, B\cup\{a,b\}, s\rangle \in [\![\mathcal{P}\,|\,\alpha]\!] \\ \text{if } g(a) = g(b) \text{ and } [f]_B = [g]_B \text{ then } r \leqslant s \end{array} \Big\}$$

We define $\langle\!\langle \mathcal{N}\,|\,\alpha \rangle\!\rangle$ from $[\![\mathcal{N}\,|\,\alpha]\!]$: $\langle\!\langle \mathcal{N}\,|\,\alpha \rangle\!\rangle = \big\{ \langle [f]_A, B, r\rangle \,\big|\, \langle f,B,r\rangle \in [\![\mathcal{N}\,|\,\alpha]\!] \big\}$ where $A = \mathbf{in}(\mathcal{N}) \cup \mathbf{out}(\mathcal{N})$.

Proposition 23 (Correctness of Flow-Network Semantics, Relativized)**.** *Let $\mathcal{N}$ be a network specification and let α be an additive aggregate objective. For every $f : \boldsymbol{A}_{in} \cup \boldsymbol{A}_{out} \cup \boldsymbol{A}_{\#} \to \mathbb{R}^+$, every $B \subseteq \boldsymbol{A}_{in} \cup \boldsymbol{A}_{out}$, and every $r \in \mathbb{R}^+$, it is the case that:*

$$\langle f,B,r\rangle \in [\![\mathcal{N}\,|\,\alpha]\!] \quad \textit{iff} \quad \begin{array}{l} f \in [\![\mathcal{N}]\!] \textit{ and } r = \alpha(f) \textit{ and} \\ \textit{for every } g \in [\![\mathcal{N}]\!],\ \textit{if } [f]_B = [g]_B \textit{ then } \alpha(g) \geqslant r. \end{array}$$

In words, for every $B \subseteq \boldsymbol{A}_{in} \cup \boldsymbol{A}_{out}$, among all feasible flows in $\mathcal{N}$ that agree on B, we include in $[\![\mathcal{N}\,|\,\alpha]\!]$ those that are α-optimal and exclude from $[\![\mathcal{N}\,|\,\alpha]\!]$ those that are not.

10 A Relativized Typing System

Let α be an additive aggregate objective, *e.g.*, one of those mentioned in Section 9. Assume α is fixed and the same throughout this section. Let $\mathcal{N}$ be a closed network specification. According to Section 7, if the judgment "$\vdash \mathcal{N} : T$" is derivable using the rules in Figure 4 and T is a valid typing, then $\mathsf{Poly}(T)$ is a set of feasible IO-flows in $\mathcal{N}$, *i.e.*, $\mathsf{Poly}(T) \subseteq \langle\!\langle \mathcal{N} \rangle\!\rangle$. And if T is principal, then in fact $\mathsf{Poly}(T) = \langle\!\langle \mathcal{N} \rangle\!\rangle$.

In this section, judgments are of the form "$\vdash \mathcal{N} : (T, \Phi)$" and derived using the rules in Figure 5. We call (T, Φ) a *relativized typing*, where T is a typing as before and Φ is an auxiliary function depending on the objective α. If T is a valid (resp. principal) typing for $\mathcal{N}$, then once more $\mathsf{Poly}(T) \subseteq \langle\!\langle \mathcal{N} \rangle\!\rangle$ (resp. $\mathsf{Poly}(T) = \langle\!\langle \mathcal{N} \rangle\!\rangle$), but now the auxiliary Φ is used to select members of $\mathsf{Poly}(T)$ that minimize α.

If this is going to work at all, Φ should not inspect the whole of $\mathcal{N}$. Instead, Φ should be defined inductively from the relativized typings for only the immediate constituent parts of $\mathcal{N}$. We first explain what the auxiliary Φ tries to achieve, and then explain how it can be defined inductively. The objective α is already defined on $[\![\mathcal{N}]\!]$, as in Section 9. We now define it on $\langle\!\langle \mathcal{N} \rangle\!\rangle$. For every $f \in \langle\!\langle \mathcal{N} \rangle\!\rangle$, let:

$$\alpha(f) = \min\big\{ \alpha(f') \,\big|\, f' \in [\![\mathcal{N}]\!] \text{ and } f' \text{ extends } f \big\}.$$

As before, let $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. Let T be a valid typing for $\mathcal{N}$, so that $\mathsf{Poly}(T) \subseteq \langle\!\langle \mathcal{N} \rangle\!\rangle$. For economy of writing, let $\mathcal{F} = \mathsf{Poly}(T)$. Relative to this T, we define the function Φ_T as follows:

$$\begin{array}{lll} \Phi_T & : & \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \to \mathscr{P}(\mathcal{F} \times \mathbb{R}^+) \\ \Phi_T(B) & = & \big\{ \langle f, r\rangle \,\big|\, f \in \mathcal{F},\ r = \alpha(f), \text{ and for every } g \in \mathcal{F}, \text{ if } [f]_B = [g]_B, \text{ then } r \leqslant \alpha(g) \big\} \end{array}$$

[5]Review footnote 3 for the meaning of "$\approx$".

where $B \in \mathscr{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$. In words, $\Phi_T(B)$ selects f provided, among all members of $\mathcal{F} \subseteq \langle\!\langle \mathcal{N} \rangle\!\rangle$ that agree with f on B, f is α-optimal – and also appends to f its α-value r for book-keeping purposes. Whenever the context makes it clear, we omit the subscript "T" from "Φ_T" and simply write "Φ".

The trick here is to define the auxiliary function Φ for $\mathcal{N}$ from the corresponding auxiliary functions for the immediate constituent parts of $\mathcal{N}$. The only non-trivial step follows the 5th and last clause in the definition of $[\![\mathcal{N} \,|\, \alpha]\!]$ in Section 9.

Definition 24 (*Valid and Principal Relativized Typings*)**.** Let (T,Φ) be a relativized typing for $\mathcal{N}$, where $\mathbf{in}(\mathcal{N}) = \mathbf{A}_{\text{in}}$ and $\mathbf{out}(\mathcal{N}) = \mathbf{A}_{\text{out}}$. We define $\mathsf{Poly}^*(T,\Phi)$ as a set of triples:

$$\mathsf{Poly}^*(T,\Phi) \;=\; \{\, \langle f,B,r\rangle \mid B \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \text{ and } \langle f,r\rangle \in \Phi(B)\,\}$$

We call this function "$\mathsf{Poly}^*()$" because of its close association with "$\mathsf{Poly}()$", as it is easy to see that:

$$\mathsf{Poly}^*(T,\Phi) \;=\; \big\{\, \langle f,B,r\rangle \;\big|\; f \in \mathsf{Poly}(T),\; B \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}},\; r = \alpha(f), \\ \text{and for all } g \in \mathsf{Poly}(T) \text{ if } [f]_B = [g]_B \text{ then } \alpha(f) \leqslant \alpha(g) \,\big\}$$

We say the relativized typing $\big(\mathcal{N} : (T,\Phi)\big)$ is *valid* iff $\mathsf{Poly}^*(T,\Phi) \subseteq \langle\!\langle \mathcal{N} \,|\, \alpha \rangle\!\rangle$, and we say it is *principal* iff $\mathsf{Poly}^*(T,\Phi) = \langle\!\langle \mathcal{N} \,|\, \alpha \rangle\!\rangle$. □

A case of particular interest is when $B = \mathbf{A}_{\text{in}}$. Suppose $\langle f,\mathbf{A}_{\text{in}},r\rangle \in \mathsf{Poly}^*(T,\Phi)$. This means that, among all feasible flows g in $\mathcal{N}$ agreeing with f on $\mathbf{A}_{\text{in}}$, f is α-optimal with $\alpha(f) = r$.

10.1 Operations on Relativized Typings

There are two different operations on relativized typings depending on how they are obtained from previously defined relativized typings. These two operations are "$(T_1,\Phi_1) \parallel (T_2,\Phi_2)$" and "$\mathsf{bind}((T,\Phi),\langle a,b\rangle)$", whose definitions are based on clauses 3 and 5 in the inductive definition of $[\![\mathcal{N} \,|\, \alpha]\!]$ in Section 9.

Let $\big(\mathcal{N}_1 : (T_1,\Phi_1)\big)$ and $\big(\mathcal{N}_2 : (T_2,\Phi_2)\big)$ be two relativized typings for two networks $\mathcal{N}_1$ and $\mathcal{N}_2$. Recall that the the four arc sets: $\mathbf{in}(\mathcal{N}_1)$, $\mathbf{out}(\mathcal{N}_1)$, $\mathbf{in}(\mathcal{N}_2)$, and $\mathbf{out}(\mathcal{N}_2)$, are pairwise disjoint. We define the relativized typing $(T,\Phi) = (T_1,\Phi_1) \parallel (T_2,\Phi_2)$ for the specification $\big(\mathcal{N}_1 \parallel \mathcal{N}_2\big)$ as follows:

- $T = (T_1 \parallel T_2)$, as defined at the beginning of Section 7.1,
- for every $B_1 \subseteq \mathbf{in}(\mathcal{N}_1) \cup \mathbf{out}(\mathcal{N}_1)$ and every $B_2 \subseteq \mathbf{in}(\mathcal{N}_2) \cup \mathbf{out}(\mathcal{N}_2)$:

$$\Phi(B_1 \cup B_2) \;=\; \big\{\, \langle (f_1 \parallel f_2), r_1 + r_2\rangle \;\big|\; \langle f_1,r_1\rangle \in \Phi_1(B_1) \text{ and } \langle f_2,r_2\rangle \in \Phi_2(B_2) \,\big\}$$

Lemma 25. *If the relativized typings $\big(\mathcal{N}_1 : (T_1,\Phi_1)\big)$ and $\big(\mathcal{N}_2 : (T_2,\Phi_2)\big)$ are principal, resp. valid, then so is the relativized typing $\big(\mathcal{N}_1 \parallel \mathcal{N}_2\big) : \big((T_1,\Phi_1) \parallel (T_2,\Phi_2)\big)$ principal, resp. valid.*

Let $\big(\mathcal{P} : (T,\Phi)\big)$ be a relativized typing for network specification $\mathcal{P}$. We define the relativized typing $(T^*,\Phi^*) = \mathsf{bind}((T,\Phi),\langle a,b\rangle)$ for the network $\textbf{bind}\,(\mathcal{P},\langle a,b\rangle)$ as follows:

- $T^* = \mathsf{bind}(T,\langle a,b\rangle)$, as defined in Section 7.1,
- for every $B \subseteq (\mathbf{in}(\mathcal{P}) \cup \mathbf{out}(\mathcal{P})) - \{a,b\}$:

$$\Phi^*(B) \;=\; \big\{\, \langle [f]_B, r\rangle \;\big|\; \langle f,r\rangle \in \Phi(B \cup \{a,b\}),\; f(a) = f(b), \text{ and for all } \langle g,s\rangle \in \Phi(B \cup \{a,b\}) \\ \text{if } g(a) = g(b) \text{ and } [f]_B = [g]_B \text{ then } r \leqslant s \,\big\}$$

Lemma 26. *If the relativized typing $\big(\mathcal{P} : (T,\Phi)\big)$ is principal, resp. valid, then so is the relativized typing $\big(\textbf{bind}\,(\mathcal{P},\langle a,b\rangle) : \mathsf{bind}((T,\Phi),\langle a,b\rangle)\big)$ principal, resp. valid.*

HOLE	$\dfrac{(X:(T,\Phi))\in\Gamma}{\Gamma\vdash {}^{i}X:({}^{i}T,{}^{i}\Phi)}$	$i\geqslant 1$ is smallest available renaming index
SMALL	$\dfrac{}{\Gamma\vdash\mathcal{A}:(T,\Phi)}$	(T,Φ) is a relativized typing for small network $\mathcal{A}$
PAR	$\dfrac{\Gamma\vdash\mathcal{N}_1:(T_1,\Phi_1)\qquad\Gamma\vdash\mathcal{N}_2:(T_2,\Phi_2)}{\Gamma\vdash(\mathcal{N}_1\,\Vert\,\mathcal{N}_2):(T_1,\Phi_1)\parallel(T_2,\Phi_2)}$	
BIND	$\dfrac{\Gamma\vdash\mathcal{N}:(T,\Phi)}{\Gamma\vdash\textbf{bind}\,(\mathcal{N},\langle a,b\rangle):\mathsf{bind}((T,\Phi),\langle a,b\rangle)}$	$\langle a,b\rangle\in\textbf{out}(\mathcal{N})\times\textbf{in}(\mathcal{N})$
LET	$\dfrac{\Gamma\vdash\mathcal{M}:(T_1,\Phi_1)\qquad\Gamma\cup\{X:(T_2,\Phi_2)\}\vdash\mathcal{N}:(T,\Phi)}{\Gamma\vdash(\,\textbf{let}\,X=\mathcal{M}\,\textbf{in}\,\mathcal{N}\,):(T,\Phi)}$	$(T_1,\Phi_1)\approx(T_2,\Phi_2)$

Figure 5: Relativized Typing Rules for Flow Networks.
The operations "$(T_1,\Phi_1)\parallel(T_2,\Phi_2)$" and "$\mathsf{bind}((T,\Phi),\langle a,b\rangle)$" are defined in Section 10.1. A derivation according to the rules is stopped from the moment a judgment $\Gamma\vdash\mathcal{N}:(T,\Phi)$ is reached such that $\mathsf{Poly}^*(T,\Phi)=\varnothing$, at which point $\mathcal{N}$ is rejected as "unsafe".

10.2 Relativized Typing Rules

Theorem 27 (Existence of Relativized Principal Typings)**.** *Let $\mathcal{N}$ be a closed network specification and (T,Φ) a relativized typing for $\mathcal{N}$ derived according to the rules in Figure 5, i.e., the judgment "$\vdash\mathcal{N}:(T,\Phi)$" is derivable according to the rules. If the relativized typing of every small network $\mathcal{A}$ in $\mathcal{N}$ is principal (resp., valid) for $\mathcal{A}$, then (T,Φ) is a principal (resp., valid) relativized typing for $\mathcal{N}$.*

11 Related and Future Work

Ours is not the only study that uses *intervals* as types and *polytopes* as typings. There were earlier attempts that heavily drew on linear algebra and polytope theory, mostly initiated by researchers who devised "types as abstract interpretations" – see [11] and references therein. However, the motivations for these earlier attempts were entirely different and applied to programming languages unrelated to our DSL. For example, polytopes were used to define "invariant safety properties", or "types" by another name, for ESTEREL – an imperative synchronous language for the development of reactive systems [15].

Apart from the difference in motivation with earlier works, there are also technical differences in the use of polytopes. Whereas earlier works consider polytopes defined by unrestricted linear constraints [12, 15], our polytopes are defined by linear constraints where every coefficient is $+1$ or -1, as implied by our Definitions 2, 3, 4, and 5. Ours are identical to the linear constraints (but not necessarily the linear objective function) that arise in the *network simplex method* [13], *i.e.*, linear programming applied to problems of network flows. There is still on-going research to improve network-simplex algorithms (*e.g.*, [22]), which will undoubtedly have a bearing on the efficiency of typing inference for our DSL.

Our polytopes-cum-typings are far more restricted than polytopes in general. Those of particular interest to us correspond to *valid* typings and *principal* typings. As of now, we do not have a characterization – algebraic or even syntactic on the shape of linear constraints – of polytopes that are *valid network typings* (or the more restrictive *principal network typings*). Such a characterization will likely guide and improve the process of typing inference.

Let $\mathcal{N}$ be a network specification, with $\mathbf{A}_{\text{in}}=\mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}}=\mathbf{out}(\mathcal{N})$. Another source of current inefficiency is that valid and principal typings for $\mathcal{N}$ tend to be "over-specified", as they unnecessarily

assign an interval-cum-type to *every* subset of $\mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$. Several examples in [19] illustrate this kind of inefficiency. This will lead us to study *partial typings* $T : \mathscr{P}(\mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}) \rightharpoonup \mathbb{R} \times \mathbb{R}$, which assign intervals to some, not necessarily all, subsets of $\mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$. Such a partial mapping T can always be extended to a total mapping $T' : \mathscr{P}(\mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}) \to \mathbb{R} \times \mathbb{R}$, in which case we write $T \subseteq T'$. We say the partial typing T is *valid* for $\mathcal{N}$ if *every* (total) typing $T' \supseteq T$ is valid for $\mathcal{N}$, and we say T is *minimal valid* for $\mathcal{N}$ if T is valid for $\mathcal{N}$ *and* for every partial typing T'' for $\mathcal{N}$ such that $T'' \subsetneq T$, *i.e.*, T'' assigns strictly fewer intervals than T, it is the case that $T \not\equiv T'$. And similarly for the definitions of partial typings that are *principal* and *minimal principal* for $\mathcal{N}$.

As alluded in the Introduction and again in Remark 9, we omitted an operational semantics of our DSL in this paper to stay clear of complexity issues arising from the associated rewrite (or reduction) rules. Among other benefits, relying on a denotational semantics allowed us to harness this complexity by performing a static analysis, via our typing theory, without carrying out a naive hole-expansion (or **let-in** elimination). We thus traded the intuitively simpler but costlier operational semantics for the more compact denotational semantics.

However, as we introduce other more complex constructs involving holes in follow-up reports (**try-in**, **mix-in**, and **letrec-in** mentioned in the Introduction and in Remark 6 of Section 3) this trade-off will diminish in importance. An operational semantics of our DSL involving these more complex hole-binders will bring it closer in line with various calculi involving *patterns* (similar to our *holes* in many ways, different in others) and where rewriting consists in eliminating *pattern-binders*. See [2, 4, 9, 10, 18] and references therein. It remains to be seen how much of the theory developed for these pattern calculi can be adapted to an operational semantics of our DSL.

References

[1] A. AuYoung, B. Chun, A. Snoeren & A. Vahdat (2004): *Resource allocation in federated distributed computing infrastructures*. In: 1st Workshop on Op Systems and Architectural Support for Ondemand IT InfraStructure. Available at `http://www.cs.ucsd.edu/~aauyoung/papers/bellagio-oasis04.pdf`.

[2] P. Baldan, C. Bertolissi, H. Cirstea & C. Kirchner (2007): *A Rewriting Calculus for Cyclic Higher-Order Term Graphs*. Math. Structures in Computer Science 17, pp. 363–406.

[3] S. Balon & G. Leduc (2006): *Dividing the Traffic Matrix to Approach Optimal Traffic Engineering*. In: 14th IEEE Int'l Conf. on Networks (ICON 2006), 2, pp. 566–571.

[4] G. Barthe, H. Cirstea, C. Kirchner & L. Liquori (2003): *Pure Patterns Type Systems*. In: Proc. 30th ACM Symp. on POPL, pp. 250–261.

[5] A. Bestavros, A. Kfoury, A. Lapets & M. Ocean (2009): *Safe Compositional Network Sketches: Tool and Use Cases*. In: IEEE Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, Wash D.C. Available at `http://www.cs.bu.edu/fac/best/res/papers/crts09.pdf`.

[6] A. Bestavros, A. Kfoury, A. Lapets & M. Ocean (2010): *Safe Compositional Network Sketches: The Formal Framework*. In: 13th ACM HSCC, Stockholm. Available at `http://www.cs.bu.edu/fac/best/res/papers/hscc10.pdf`.

[7] J.-Y. Le Boudec & P. Thiran (2004): *Network Calculus*. Springer Verlag. LNCS 2050.

[8] R. Buyya, D. Abramson & J. Giddy (2000): *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*. In: HPC ASIA 2000. Available at `http://www.gridbus.org/papers/nimrodg.pdf`.

[9] H. Cirstea, C. Kirchner & L. Liquori (2004): *Rewriting Calculus with(out) Types*. Electronic Notes in Theoretical Computer Science 71, pp. 3–19.

[10] H. Cirstea, L. Liquori & B. Wack (2003): *Rewriting Calculus with Fixpoints: Untyped and First-order Systems*. In: Post-proceedings of TYPES, LNCS, Springer, pp. 147–161.

[11] P. Cousot (1997): *Types as Abstract Interpretations, invited paper*. In: *Proc. of 24th ACM Symp. on Principles of Programming Languages*, Paris, pp. 316–331.

[12] P. Cousot & N. Halbwachs (1978): *Automatic Discovery of Linear Restraints Among Variables of a Program*. In: *Proc. 5th ACM Symp. on POPL*, Tucson, pp. 84–96.

[13] W. H. Cunningham (1979): *Theoretical Properties of the Network Simplex Method*. *Mathematics of Operations Research* 4(2), pp. 196–208.

[14] J. Gomoluch & M. Schroeder (2004): *Performance evaluation of market-based resource allocation for Grid computing*. *Concurrency and Computation: Practice and Experience* 16(5), pp. 469–475.

[15] N. Halbwachs (1993): *Delay Analysis in Synchronous Programs*. In: *Fifth Conference on Computer-Aided Verification*, LNCS 697, Springer Verlag, Elounda (Greece).

[16] V. Ishakian, A. Bestavros & A. Kfoury (2010): *A Type-Theoretic Framework for Efficient and Safe Colocation of Periodic Real-time Systems*. In: *Int'l Conf on Embedded and Real-Time Computing Systems and Applications (RTSCA'10)*, Macau, China. Available at http://www.cs.bu.edu/fac/best/res/papers/rtsca10.pdf.

[17] V. Ishakian, R. Sweha, J. Londono & A. Bestavros (2010): *Colocation as a Service: Strategic and Operational Services for Cloud Colocation*. In: *Int'l Symp on Network Computing and Applications (NCA'10)*, Cambridge, MA. Available at http://www.cs.bu.edu/fac/best/res/papers/nca10.pdf.

[18] C. Barry Jay & D. Kesner (2006): *Pure Pattern Calculus*. In: *European Symposium on Programming*, pp. 100–114.

[19] A. Kfoury (2011): *A Domain-Specific Language for Incremental and Modular Design of Large-Scale Verifiably-Safe Flow Networks (Part 1)*. Technical Report BUCS-TR-2011-011, CS Dept, Boston Univ.

[20] E. Knightly & H. Zhang (1997): *D-BIND: an accurate traffic model for providing QoS guarantees to VBR traffic*. *IEEE/ACM Transactions on Networking* 5, pp. 219–231.

[21] C. L. Liu & James W. Layland (1973): *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*. *Journal of the ACM* 20(1), pp. 46–61, doi:http://doi.acm.org/10.1145/321738.321743.

[22] H. Rashidi & E.P.K. Tsang (2009): *An Efficient Extension of Network Simplex Algorithm*. *Journal of Industrial Engineering* 2, pp. 1–9.

[23] J. Regehr & J.A. Stankovic (2001): *HLS: A Framework for Composing Soft Real-Time Schedulers*. In: *22nd IEEE Real-Time Systems Symposium (RTSS '01)*, IEEE Comp Soc, Washington, DC, USA, p. 3.

[24] I. Shin & I. Lee (2003): *Periodic Resource Model for Compositional Real-Time Guarantees*. In: *24th IEEE International Real-Time Systems Symposium (RTSS '03)*, IEEE Comp Soc, Washington, DC, USA, p. 2.

[25] N. Soule, A. Bestavros, A. Kfoury & A. Lapets (2011): *Safe Compositional Equation-based Modeling of Constrained Flow Networks*. In: *Proc. of 4th Int'l Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Zürich.

[26] P. Thiran, J. Boudec & F. Worm (2001): *Network calculus applied to optimal multimedia smoothing*. In: *Proceedings of IEEE INFOCOM*.

[27] R. Wolski, J.S. Plank, J. Brevik & T. Bryan (2001): *G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid*. In: *15th Int'l Parallel & Distributed Processing Symposium (IPDPS '01)*, IEEE Comp Soc, Washington, DC, USA, p. 46.