

A Domain-Specific Language for the Incremental and Modular Design of Large-Scale Verifiably-Safe Flow Networks (Part 1)

Assaf Kfoury

May 19, 2011

Contents

1	Introduction	2
2	Preliminary Definitions	3
2.1	Flow Conservation, Capacity Constraints, Type Satisfaction	5
3	DSL for Incremental and Modular Design of Flow Networks (Untyped)	6
3.1	Derived Constructors	7
3.2	Well-Formed Network Specifications	9
4	Semantics of Flow Networks	12
4.1	Flow Conservation, Capacity Constraints, Type Satisfaction (Continued)	15
5	Typings Are Polytopes	16
5.1	Uniqueness and Redundancy in Typings	17
5.2	Valid Typings and Principal Typings	21
6	Other Properties and Open Problems of Network Typings	22
7	Inferring Typings for Small Networks	24
8	A Typing System	26
8.1	Operations on Typings	27
8.2	Typing Rules	29
9	Inferring Typings for Flow Networks in General	30
10	Semantics of Flow Networks Relative to Objective Functions	34
11	A Relativized Typing System	39
11.1	Operations on Relativized Typings	40
11.2	Relativized Typing Rules	41

Abstract

Flow networks are inductively defined, assembled from *small networks* or *modules* to produce arbitrarily large ones, with interchangeable functionally-equivalent parts. We carry out this induction formally using a *domain-specific language* (DSL). Associated with our DSL is a *typing system* (or *static semantics*), a system of formal annotations that enforce desirable properties of flow networks as *invariants* across their interfaces. 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. We carry out this in two ways, as a *denotational* semantics and as an *operational* (or *reduction*) semantics.

1 Introduction

What we call *flow networks* are inductively defined, assembled from *small networks* or *modules* to produce arbitrarily large ones, with interchangeable functionally-equivalent parts. We carry out this induction formally using a *domain-specific language* (DSL). Our small networks or modules are “small” only as the building blocks in this inductive definition (there is no limit on their size).

Our DSL provides two primitive constructors, one of the form $(\mathcal{M}_1 \parallel \mathcal{M}_2)$ and the other **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 two primitive constructors, we define others as *derived* constructors and according to need. Our DSL allows for the presence of *holes* in network specifications by means of a constructor 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, depends on the invariant properties that typings are formulated to enforce. We also consider variations of this construction, of the form:

- (1) **let** $X \in \{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ **in** \mathcal{N}
- (2) **try** $X \in \{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ **in** \mathcal{N}
- (3) **mix** $X \in \{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ **in** \mathcal{N}

The specification in (1) informally says “every \mathcal{M}_i may be safely placed in all the occurrences of X in \mathcal{N} ”, the one in (2) says “at least one \mathcal{M}_i may be safely placed in all the occurrences of X in \mathcal{N} ”, and the one in (3) says “every mix of several \mathcal{M}_i ’s may be selected and safely placed in the occurrences of X in \mathcal{N} , generally placing different \mathcal{M}_i ’s from that mix in different occurrences”. A rigorous definition of these three meanings is entrusted to an appropriate formal semantics in each case.

Together with our DSL, we define a *type theory* (or *static semantics*), which is a system of formal annotations that enforce desirable properties of flow networks as *invariants* across their interfaces, *i.e.*, 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. We carry out this in two ways, as a *denotational* semantics and 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.

We can prove the equivalence (in a sense that can be made precise) of the two approaches. However, 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. We can also prove the soundness of the typing system relative to the thus-defined formal semantics (“a type-safe network construction guarantees that flows through the network satisfy the invariants properties enforced by types”).

Organization of the report. 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** and **mix-in** to a follow-up report.

The formal semantics of flow networks are introduced in Section 4 and a corresponding type theory 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*.¹ We only include an examination of modular typing inference in this report, leaving its (more elaborate) fully-compositional version to a follow-up report.

The remaining sections in this report expand on the fundamentals laid out in the first five sections. In Section 6, we present several open problems and discuss their relevance for the further development of our DSL, its semantics and associated type theory. Sections 7 to 11 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 provided they additionally satisfy appropriately defined objective functions (introduced in Section 10).

This report is the first in a three-part sequence. In the second, we deal with fully-compositional typing inference. In the third, we augment our DSL with the **try-in** and **mix-in** constructors, and then examine the resulting formal semantics and typing theory.

Acknowledgement. Our work in this report is a small fraction of a collective effort involving several people, under the umbrella of the **iBench Initiative** at Boston University. A list of participants, former participants, and other research activities, can be found at <https://sites.google.com/site/ibenchbu/>

The DSL in this report, 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 [BKLO09, BKLO10]. 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 $L : \mathbf{A} \rightarrow \mathbb{R}^+$ and an upper-bound $U : \mathbf{A} \rightarrow \mathbb{R}^+$ satisfying the conditions $L(a) \leq U(a)$ 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 $\text{head}(a)$ and $\text{tail}(a)$, with the understanding that flow moves from $\text{tail}(a)$ to $\text{head}(a)$. The set \mathbf{A} of arcs is the disjoint union of three sets: the set $\mathbf{A}_\#$ of internal arcs, the set \mathbf{A}_{in} of input arcs, and the set \mathbf{A}_{out} of output arcs:

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

The tail of any input arc is not attached to any node, and the head of an output arc is not attached to any node.

We do not assume that \mathcal{A} is connected as a directed graph – an assumption often made in studies of network flows, which is sensible when there is only one input arc (or one “source node”) and only one output arc (or

¹We add the qualifier “fully” to distinguish our notion of compositionality from similar, but different, notions in other areas of computer science. Adding to the imprecision of the word, “compositional” in the literature is sometimes used in the more restrictive sense of “modular” in our sense.

one “sink node”). We assume that $\mathbf{N} \neq \emptyset$, *i.e.*, there is at least one node in \mathbf{N} , without which there would be no input arc, no output arc, and nothing to say.

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

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

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

where $\mathcal{P}(\cdot)$ is the power-set operator, *i.e.*, $\mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) = \{A \mid A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}\}$. 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.

If T is a typing, we write $[T]_{\text{in}}$ and $[T]_{\text{out}}$ to denote the restriction of T to input arcs and output arcs, respectively. Specifically,

$$[T]_{\text{in}}(A) = \begin{cases} T(A) & \text{if } A \subseteq \mathbf{A}_{\text{in}}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

and similarly for the definition of $[T]_{\text{out}}$.

Remark 1. In the presence of a lower-bound function L and an upper-bound function U , we do not need to distinguish some nodes in \mathbf{N} as *producers* and some others as *consumers*. For example, if n is a node that produces an amount $r \in \mathbb{R}^+$, we can instead introduce 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 can instead introduce a new output arc a' exiting n' with $L(a') = U(a') = r'$. For reference in Remark 2, call these new a and a' *special κ -arcs*. The resulting network \mathcal{A}' is equivalent to the original \mathcal{A} , in the sense that any feasible flow in the first induces a feasible flow in the second, and vice-versa. \square

Remark 2. The analysis to follow is restricted to a single commodity. It is straightforward to generalize it to several commodities from a finite set K of commodities. This requires a separate definition of a flow function $f_\kappa : \mathbf{A} \rightarrow \mathbb{R}^+$ for each commodity $\kappa \in K$. An equation for flow conservation (expressed by (1) below) has to be set up for each commodity separately, but the inequality constraints (expressed by (2) below) must be satisfied by the sum of all the commodity flows together.

If we want some nodes as producers or consumers of a particular commodity $\kappa \in K$, we need to introduce lower-bound and upper-bound functions L_κ and U_κ such that $L_\kappa(a) = L(a)$ and $U_\kappa(a) = U(a)$ on every arc a that is not a special κ -arc, and $L(a) = U(a) = L_{\kappa'}(a) = U_{\kappa'}(a) = 0$ for every special κ' -arc a , where $\kappa' \neq \kappa$. This will force special κ -arcs to carry flow of commodity κ only.

In the presence of several commodities, our framework is general enough for an examination of what is called the *demand matrix* in traffic engineering. The demand matrix D for a network with a set \mathbf{N} of nodes is a square matrix of size $|\mathbf{N}| \times |\mathbf{N}|$ where the entry $D(n, n') = r \in \mathbb{R}^+$ denotes the amount to be sent from node n to node n' . If $r \neq 0$, we identify a commodity (or a kind) called $\kappa_{(n, n')}$ with the pair (n, n') and make node n a producer of an amount r of kind $\kappa_{(n, n')}$ and node n' a consumer of the same amount r of kind $\kappa_{(n, n')}$. \square

Remark 3. We do not disallow the special cases when $\mathbf{A}_{\text{in}} = \emptyset$, or $\mathbf{A}_{\text{out}} = \emptyset$, or both, because they may result from some of our later constructions. However, by themselves, these cases are quickly analyzed.

²Our notion of a “typing” as an assignment of types to the members of a powerset is different from a notion by the same name 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 that includes a type for every variable occurring free in M .

Thus, if $\mathbf{A}_{\text{in}} = \emptyset$ and $\mathbf{A}_{\text{out}} \neq \emptyset$, a feasible flow f in \mathcal{A} must assign 0 to every output arc – unless there is an output arc a with non-zero lower bound $L(a) \neq 0$, in which case there is no feasible flow in \mathcal{A} .

Similarly, if $\mathbf{A}_{\text{in}} \neq \emptyset$ and $\mathbf{A}_{\text{out}} = \emptyset$, then a feasible flow f in \mathcal{A} must assign 0 to every input arc – unless there is an input arc a with non-zero lower bound $L(a) \neq 0$, in which case there is no feasible flow in \mathcal{A} .

And if both $\mathbf{A}_{\text{in}} = \mathbf{A}_{\text{out}} = \emptyset$, the network \mathcal{A} is “totally closed” (in the terminology of Section 3) and cannot interact with any other network. If \mathcal{A} is a small module, which cannot thus be hooked to any other, our typing theory has nothing to say about \mathcal{A} . If \mathcal{A} is the result of an inductive construction – and is “totally closed” because all output arcs have been looped back to enter input arcs – then our typing theory guarantees that \mathcal{A} ’s internal working respects the invariant properties that the types were formulated to express. \square

2.1 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 4, 5, 6, and 7.

Definition 4 (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 :

$$\sum f(A) = \sum \{ f(a) \mid a \in A \}$$

By convention, $\sum \emptyset = 0$. 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 following linear equation:

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

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

Definition 5 (Capacity Constraints). A flow f satisfies the capacity constraints at arc $a \in \mathbf{A}$ if:

$$(2) \quad L(a) \leq f(a) \leq U(a)$$

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

Definition 6 (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. \square

Definition 7 (Type Satisfaction). Let $T : \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$ be a typing for the small network \mathcal{A} . We say the flow f *satisfies* T if, for every $A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$ with $T(A) = [r, r']$, it is the case:

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

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

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 compared 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}_{in} and \mathbf{A}_{out} are finite sets of input and output arcs. We also write $X = (X, \mathbf{A}_{\text{in}}, \mathbf{A}_{\text{out}})$ where, without confusion, X is used as the *name* of the hole denoted by the same letter. The idea of a hole X is that it is a place holder where networks can be inserted or removed, provided the *matching-dimensions* condition (defined 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 will later depend on the invariant properties that typings are formulated to represent. In such an expression, we call the X to the immediate left of “ ϵ ” 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 $\mathbf{A} = \mathbf{A}_{\#} \cup \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, let $\text{in}(\mathcal{A}) = \mathbf{A}_{\text{in}}$, $\text{out}(\mathcal{A}) = \mathbf{A}_{\text{out}}$, and $\#(\mathcal{A}) = \mathbf{A}_{\#}$. Similarly, if $X = (\mathbf{A}_{\text{in}}, \mathbf{A}_{\text{out}})$ is a hole, let $\text{in}(X) = \mathbf{A}_{\text{in}}$, $\text{out}(X) = \mathbf{A}_{\text{out}}$, and $\#(X) = \emptyset$. 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**, and **bind**:

$\mathcal{A}, \mathcal{B}, \mathcal{C}$	\in SMALLNETWORKS	
X, Y, Z	\in HOLENAMES	
$\mathcal{M}, \mathcal{N}, \mathcal{P}$	\in NETWORKS	
	$::=$	\mathcal{A} small network name
		X hole name
		$\mathcal{M} \parallel \mathcal{N}$ parallel connection
		let $X = \mathcal{M}$ in \mathcal{N} let-binding of hole X , $n \geq 1$
		bind $(\mathcal{N}, \langle a, b \rangle)$ bind head(a) to tail(b), where $\langle a, b \rangle \in \text{out}(\mathcal{N}) \times \text{in}(\mathcal{N})$

where $\text{in}(\mathcal{N})$ to $\text{out}(\mathcal{N})$ are the input and output arcs of \mathcal{N} , defined simultaneously with the set $\#(\mathcal{N})$ of internal arcs, by induction:

- If \mathcal{N} is the name of small network \mathcal{A} , then
 $\text{in}(\mathcal{N}) = \text{in}(\mathcal{A})$, $\text{out}(\mathcal{N}) = \text{out}(\mathcal{A})$, and $\#(\mathcal{N}) = \#(\mathcal{A})$.
- If \mathcal{N} is the name of hole X , then
 $\text{in}(\mathcal{N}) = \text{in}(X)$, $\text{out}(\mathcal{N}) = \text{out}(X)$, and $\#(\mathcal{N}) = \emptyset$.
- If $\mathcal{N} = (\mathcal{M} \parallel \mathcal{M}')$, then
 $\text{in}(\mathcal{N}) = \text{in}(\mathcal{M}) \cup \text{in}(\mathcal{M}')$, $\text{out}(\mathcal{N}) = \text{out}(\mathcal{M}) \cup \text{out}(\mathcal{M}')$, and $\#(\mathcal{N}) = \#(\mathcal{M}) \cup \#(\mathcal{M}')$.
- If $\mathcal{N} = (\text{let } X = \mathcal{M} \text{ in } \mathcal{N}')$, then
 $\text{in}(\mathcal{N}) = \text{in}(\mathcal{N}')$, $\text{out}(\mathcal{N}) = \text{out}(\mathcal{N}')$, and $\#(\mathcal{N}) = \#(\mathcal{N}') \cup \#(\mathcal{M})$.
- If $\mathcal{N} = \text{bind}(\mathcal{N}', \langle a, b \rangle)$, then
 $\text{in}(\mathcal{N}) = \text{in}(\mathcal{N}') - \{b\}$, $\text{out}(\mathcal{N}) = \text{out}(\mathcal{N}') - \{a\}$, and $\#(\mathcal{N}) = \#(\mathcal{N}') \cup \{a\}$ with $\text{head}(a) := \text{tail}(b)$.

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 $\mathbf{in}(\mathcal{N}) = \mathbf{out}(\mathcal{N}) = \emptyset$, *i.e.*, \mathcal{N} has no input arcs and no output arcs.

Remark 8. A network specification \mathcal{N} , as defined by the BNF above, does not introduce lower-bound and upper-bound capacities on arcs. \mathcal{N} only defines a topology of a large network, starting from a collection of small networks. From the capacities assigned to the small networks' arcs, our typing theory will attempt to infer typings for all the well-formed subparts (or subexpressions) of \mathcal{N} and for \mathcal{N} itself. If it succeeds to do this inference, the typings will certify that the construction of every larger part from smaller parts respects the invariant properties we wish to impart to all of \mathcal{N} .

Among invariant properties, we will want, at a minimum, that if there are feasible flows in the smaller parts, then there are feasible flows in the larger parts. There is one construction which already illustrates this idea. In a subexpression of the form $\mathbf{bind}(\mathcal{N}, \langle a, b \rangle)$, if the lower-bound capacity of arc a (or b , resp.) is strictly larger than the upper-bound capacity of arc b (or a , resp.), then there is no feasible flow in $\mathbf{bind}(\mathcal{N}, \langle a, b \rangle)$ and our typing theory will not allow this “unsafe” binding, *i.e.*, connecting the head of a to the tail of b . \square

3.1 Derived Constructors

From the three constructors already introduced, namely: \parallel , **let**, and **bind**, we can define several other constructors. The latter will be called *derived constructors* to distinguish them from the three earlier *primitive constructors*. Below, we present four of these derived constructors precisely, and mention several others in Remark 10. Our four derived constructors will be 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:

- **bind** (\mathcal{N}, θ)
- **conn**($\mathcal{N}_1, \mathcal{N}_2, \theta$)
- $\mathcal{N}_1 \oplus \mathcal{N}_2$
- **let** $X \in \{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ **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 $\theta \subseteq_{1-1} \mathbf{out}(\mathcal{N}) \times \mathbf{in}(\mathcal{N})$ to denote a partial one-one map from $\mathbf{out}(\mathcal{N})$ to $\mathbf{in}(\mathcal{N})$. We may write the entries in θ explicitly, as in:

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

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

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 $\langle a, b \rangle \in \mathbf{out}(\mathcal{N}) \times \mathbf{in}(\mathcal{N})$. The expression $\mathbf{bind}(\mathcal{N}, \theta)$ can be expanded as follows:

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

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 (omitted) shows that the order in which we connect arc heads to arc tails does not matter as far as our typing theory is concerned, *i.e.*, it infers the same typing (if it exists) for the whole construction regardless of what that order is. Remark 9 elaborates this last point.

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 $\theta \subseteq_{1-1} \mathbf{out}(\mathcal{N}_1) \times \mathbf{in}(\mathcal{N}_2)$. We expand the expression $\mathbf{conn}(\mathcal{N}_1, \mathcal{N}_2, \theta)$ as follows:

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

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 $m \geq 1$ 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 those 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. In this case we write $(\mathcal{N}_1 \oplus \mathcal{N}_2)$, which can be expanded as follows:

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

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, it turns out that \oplus is associative as far as our typing theory is concerned, *i.e.*, it infers the same typing for $\mathcal{N}_1 \oplus (\mathcal{N}_2 \oplus \mathcal{N}_3)$ and $(\mathcal{N}_1 \oplus \mathcal{N}_2) \oplus \mathcal{N}_3$, and we will simply write $\mathcal{N}_1 \oplus \mathcal{N}_2 \oplus \mathcal{N}_3$. (See Remark 9 for an elaboration of this.)

A fourth and last derived constructor is a generalization of **let** which can be expanded into several nested let-bindings:

$$\left(\mathbf{let} X \in \{\mathcal{M}_1, \dots, \mathcal{M}_n\} \mathbf{in} \mathcal{N} \right) \implies \left(\mathbf{let} X_1 = \mathcal{M}_1 \mathbf{in} \left(\dots \left(\mathbf{let} X_n = \mathcal{M}_n \mathbf{in} (\mathcal{N}_1 \parallel \dots \parallel \mathcal{N}_n) \right) \dots \right) \right)$$

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 $1 \leq i \leq n$. 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} .

We will use these 4 derived constructors in examples, but when setting up the typing rules for networks in general, we will revert back to the primitive constructors.

Remark 9. If we consider graphical representations of constructions such as **bind** (\mathcal{N}, θ) and $\mathcal{N}_1 \oplus \mathcal{N}_2 \oplus \mathcal{N}_3$, then the order in which we connect the arcs in the graphs does not matter, obviously. But we will invoke graphical representations only informally. To formally translate our network specifications into graphical representations in some unique normal form – which requires not only expanding all derived constructors but also, more challengingly, introducing formal rules to reduce all let-bindings – is the basis of an *operational* (or *reduction*) approach to the semantics of network specifications. However, this is something we try to avoid as much as we can, for reasons we further elaborate in Remark 12 below.

Whenever we need to invoke the semantics of network specifications, we use the *denotational* definition presented in Section 4 in preference to an *operational* definition. \square

Remark 10. Other derived constructors can be defined according to need in applications. We sketch a few. For example, an obvious generalization of \oplus will cascade the same network \mathcal{N} some $n \geq 1$ times, for which we may write $\oplus(\mathcal{N}, n)$. A condition for well-formedness in this case is that \mathcal{N} 's input and output dimensions must be equal.

Another derived constructor may be **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 and which requires, for well-formedness, 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 **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

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

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

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

which we can take to mean that a mixture of *several* \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.

We leave the treatment of the constructors **try** and **mix** to a follow-up report. An informal understanding of how these two differ from the constructor **let** can be gleaned from Example 13. \square

3.2 Well-Formed Network Specifications

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. Also, although they could be easily incorporated into our inductive definition, more general than BNF style, they would obscure the relatively simple structure of our network specifications.

Matching dimensions of input/output arcs

Let \mathcal{M} be a network specification. We assume there is a fixed ordering of the entries in $\mathbf{in}(\mathcal{M})$ and $\mathbf{out}(\mathcal{M})$. If we need to refer to both together, we agree that the arcs in $\mathbf{in}(\mathcal{M})$ are listed before those in $\mathbf{out}(\mathcal{M})$:

$\dim_{\mathbf{in}}(\mathcal{M})$ is $\mathbf{in}(\mathcal{M})$ as an ordered set – *input dimension* of \mathcal{M} .

$\dim_{\mathbf{out}}(\mathcal{M})$ is $\mathbf{out}(\mathcal{M})$ as an ordered set – *output dimension* of \mathcal{M} .

$\dim(\mathcal{M}) = \dim_{\mathbf{in}}(\mathcal{M}) \cdot \dim_{\mathbf{out}}(\mathcal{M})$ is $\mathbf{in}(\mathcal{M}) \cup \mathbf{out}(\mathcal{M})$ as an ordered set – *I/O dimension* of \mathcal{M} .

In the let-binding of a hole X we have to make sure that the network considered for insertion in X has the same number of input arcs, the same number of output arcs, and both are ordered in the same way. More precisely, an expression of the form:

$$\mathbf{let} \ X = \mathcal{M} \ \mathbf{in} \ \mathcal{N}$$

is *well-formed* provided:

$$\dim_{\mathbf{in}}(X) \approx \dim_{\mathbf{in}}(\mathcal{M}) \quad \text{and} \quad \dim_{\mathbf{out}}(X) \approx \dim_{\mathbf{out}}(\mathcal{M})$$

where “ \approx ” indicates that the first arc, second arc, etc., in X correspond to the first arc, second arc, etc., in \mathcal{M} . Keep in mind that arcs are named differently in X and in \mathcal{M} , which is why we write “ \approx ” instead of “ $=$ ”. If the preceding condition is satisfied, we will say that X and \mathcal{M} have *similar* input and output dimensions. Thus, when we place \mathcal{M} in hole X , we connect the designated first arc, second arc, etc., in X to the designated first arc, second arc, etc., in \mathcal{M} , respectively.

Moreover, if there are several, say $k \geq 2$, occurrences of X in \mathcal{N} , we want each of the k copies of X to have its distinct set of input arcs and distinct set of output arcs, as we discuss next.

Unique arc naming

We need to guarantee that, in the specification of a network \mathcal{N} , no arc name refers to two different arcs. This is needed in order to avoid some ambiguities later. This condition is not enforced by the BNF definition, but we can enforce it by appropriate “isomorphic renaming”, *i.e.*, by renaming arc names in order to avoid a single name for several arcs without changing the topology of the network, as we explain next.

We first define the *outer scope* and *inner scope* of a let-binding for a hole X in a network specification \mathcal{N} : the inner scope is the part of \mathcal{N} where all the bound occurrences of X are mentioned, here indicated by an underbrace:

$$\mathcal{N} = \underbrace{\dots}_{\text{outer scope}} \left(\text{let } X = \underbrace{\dots}_{\text{outer scope}} \text{ in } \underbrace{\dots X \dots X \dots}_{\text{inner scope of } X} \right) \underbrace{\dots}_{\text{outer scope}}$$

Inner scopes may be disjoint, as in:

$$\mathcal{N} = \dots \left(\text{let } X = \dots \text{ in } \underbrace{\dots X \dots X \dots}_{\text{inner scope of } X} \right) \dots \left(\text{let } Y = \dots \text{ in } \underbrace{\dots Y \dots Y \dots}_{\text{inner scope of } Y} \right) \dots$$

and they may be nested, as in:

$$\mathcal{N} = \dots \left(\text{let } X = \dots \text{ in } \dots X \dots \underbrace{\left(\text{let } Y = \dots \text{ in } \underbrace{\dots Y \dots X \dots Y \dots}_{\text{inner scope of } Y} \right) \dots X \dots} \right) \dots$$

inner scope of X

We need to distinguish the arcs of the different copies of the same hole X *within the inner scope of X* . Thus, if we use $k \geq 2$ copies of X within the same scope, we rename their arcs so that each copy has its own set of arcs. We write ${}^1X, \dots, {}^kX$ to refer to these k copies of X . However, we do *not* rename the corresponding binding occurrence of X . Thus, the two last of the three schematic representations above should be written as:

$$\mathcal{N} = \dots \left(\text{let } X = \dots \text{ in } \underbrace{\dots {}^1X \dots {}^2X \dots}_{\text{inner scope of } X} \right) \dots \left(\text{let } Y = \dots \text{ in } \underbrace{\dots {}^1Y \dots {}^2Y \dots}_{\text{inner scope of } Y} \right) \dots$$

$$\mathcal{N} = \dots \left(\text{let } X = \dots \text{ in } \underbrace{\dots {}^1X \dots \left(\text{let } Y = \dots \text{ in } \underbrace{\dots {}^1Y \dots {}^2X \dots {}^2Y \dots}_{\text{inner scope of } Y} \right) \dots {}^3X \dots}_{\text{inner scope of } X} \right) \dots$$

As we also keep track of the fact that ${}^1X, \dots, {}^kX$ are all copies of X , there will be no ambiguity about which holes in \mathcal{N} this binding occurrence of X refers to.

In addition to the preceding, the *unique arc-naming* condition requires that, if a network specification \mathcal{N} mentions $k \geq 2$ copies of the same small network \mathcal{A} , then each copy has its own separate set of arc names. Put differently, \mathcal{N} mentions a small network \mathcal{A} at most once, though it may mention several other small networks that are all isomorphic to \mathcal{A} .

One binding-occurrence for every hole X

For well-formedness we also require that, for every hole X , there is at most one let-binding for X , *i.e.*, there is at most one binding occurrence of X . This condition disallows specifications \mathcal{N} that are of the form:

$$\mathcal{N} = \dots \left(\text{let } X = \dots \text{ in } \underbrace{\dots X \dots X \dots}_{\text{inner scope of } X} \right) \dots \left(\text{let } X = \dots \text{ in } \underbrace{\dots X \dots X \dots}_{\text{inner scope of } X} \right) \dots$$

where there are two let-bindings of X for two disjoint scopes. And it disallows specifications \mathcal{N} of the form:

$$\mathcal{N} = \dots \left(\text{let } X = \dots \text{ in } \underbrace{\dots X \dots \left(\text{let } X = \dots \text{ in } \underbrace{\dots X \dots X \dots}_{\text{inner scope of } X} \right) \dots X \dots}_{\text{inner scope of } X} \right) \dots$$

where there are two let-bindings of X for two nested scopes.

We are mostly interested in analyzing *closed* network specifications and determining their safety properties. Observe that, for a closed network specification \mathcal{N} , the *one binding-occurrence* condition disallows the presence of subexpressions in \mathcal{N} of the form:

$$\dots (\mathbf{let} \ X = \dots \underset{\uparrow}{X} \dots \ \mathbf{in} \ \underbrace{\dots X \dots X \dots}) \dots$$

where the X indicated by the upward arrow is outside the inner scope of the binding occurrence of X .

Remark 11. Of the three conditions for well-formedness, only the *matching-dimensions* is essential for setting up the topology correctly of large networks from their smaller parts.

The other two conditions, the *unique arc-naming* and the *one binding-occurrence*, are introduced for the purposes of the typing theory later; and of these two, the *one binding-occurrence* can be omitted, but at the cost of unduly complicating things. \square

Remark 12. It is possible to define the notion of a network specification in *normal form*, one in which every let-binding has been reduced. But this raises several technical complications. For example, because let-bindings can be nested to any depth, the reduction of let-bindings leads to an exponential explosion in the size of normal forms.

Moreover, reduction of a let-binding “ $\mathbf{let} \ X = \mathcal{M} \ \mathbf{in} \ \mathcal{N}$ ” violates the *unique arc-naming* condition, whenever X occurs more than once in \mathcal{N} . It also violates the *one binding-occurrence* condition, whenever the \mathcal{M} to be substituted for X contains itself a let-binding and X occurs more than once in \mathcal{N} . Reduction rules can be modified so these two conditions are not violated, but again at the price of technical complications. \square

Example 13. We illustrate several notions with an extended example. We use one hole X , and 4 small networks: \mathbf{F} (“fork”), \mathbf{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 \mathbf{F} , \mathbf{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 \mathbf{F} , \mathbf{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:

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

where

$$\begin{aligned} \theta_1 &= \{\langle c_2, {}^1e_1 \rangle, \langle c_3, {}^1e_2 \rangle\} \\ \theta_2 &= \{\langle {}^1e_3, {}^2e_1 \rangle, \langle {}^1e_4, {}^2e_2 \rangle\} \\ \theta_3 &= \{\langle {}^2e_3, d_1 \rangle, \langle {}^2d_4, d_2 \rangle\} \end{aligned}$$

We wrote \mathcal{N} above using some of the derived constructors introduced in Section 3.1. We can write \mathcal{N} even more succinctly by noting that:

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

According to Section 3.1, we can write more simply:

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

with now $\mathbf{in}(\mathcal{N}) = \{c_1\}$ and $\mathbf{out}(\mathcal{N}) = \{d_3\}$.

The specification \mathcal{N} says that small network \mathcal{A} or small network \mathcal{B} can be selected for insertion wherever hole X occurs. Though we do not define the reduction of let-bindings formally, \mathcal{N} can be viewed as representing two different network configurations:

$$\mathcal{N}_1 = \mathbf{F} \oplus^1 \mathcal{A} \oplus^2 \mathcal{A} \oplus \mathbf{M}$$

$$\mathcal{N}_2 = \mathbf{F} \oplus^1 \mathcal{B} \oplus^2 \mathcal{B} \oplus \mathbf{M}$$

We can say nothing here about properties, such as safety, being satisfied or violated by these two configurations. The semantics of our **let** constructor later will be equivalent to requiring that both configurations be “safe” to use. By contrast, the constructor **try** mentioned in Remark 10 requires only \mathcal{N}_1 or \mathcal{N}_2 , but not necessarily both, to be safe, and the constructor **mix** additionally requires:

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

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

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

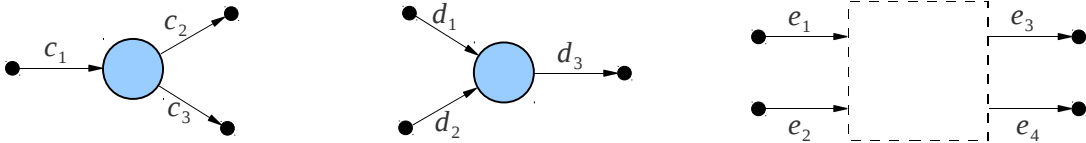


Figure 1: Small network \mathbf{F} (on the left), small network \mathbf{M} (in the middle), and hole X (on the right), in Example 13.

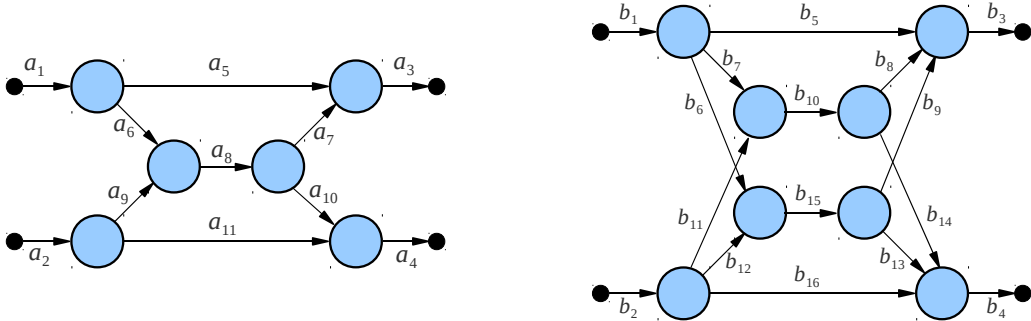


Figure 2: Small networks \mathcal{A} (on the left) and \mathcal{B} (on the right) in Example 13.

4 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 occurrence iX of a hole X also has its own separate set of arc names, where $i \geq 1$ is a renaming index. With every small network \mathcal{A} , we associate two sets of functions, its *full semantics* $\llbracket \mathcal{A} \rrbracket$ and its *IO-semantics* $\langle\langle \mathcal{A} \rangle\rangle$. Let

$$\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{A}), \quad \mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{A}), \quad \mathbf{A}_{\#} = \#(\mathcal{A}), \quad \text{and} \quad \mathbf{A} = \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \cup \mathbf{A}_{\#}.$$

The sets $\llbracket \mathcal{A} \rrbracket$ and $\langle\langle \mathcal{A} \rangle\rangle$ are defined thus:

$$\begin{aligned} \llbracket \mathcal{A} \rrbracket &= \{ f : \mathbf{A} \rightarrow \mathbb{R}^+ \mid f \text{ is a feasible flow in } \mathcal{A} \} \\ \langle\langle \mathcal{A} \rangle\rangle &= \{ f : \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \rightarrow \mathbb{R}^+ \mid f \text{ can be extended to a feasible flow } f' \text{ in } \mathcal{A} \} \end{aligned}$$

Both $\llbracket \mathcal{A} \rrbracket$ and $\langle\langle \mathcal{A} \rangle\rangle$ are uniquely defined.

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

$$\llbracket X \rrbracket = \langle\langle X \rangle\rangle \subseteq \{ f : \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \rightarrow \mathbb{R}^+ \mid r \leq \sum f(\mathbf{A}_{\text{in}}) = \sum f(\mathbf{A}_{\text{out}}) \leq r' \}$$

for some real numbers $0 \leq r \leq r'$. We do not define $\llbracket X \rrbracket = \langle\langle X \rangle\rangle$ further. In contrast to the uniquely defined full semantics and IO-semantics of a small network, there are clearly infinitely many $\llbracket X \rrbracket = \langle\langle X \rangle\rangle$ for the same X , although only one will work, namely, the one satisfying the requirement in clause 4 below. By well-formedness, there is at most one let-binding in \mathcal{N} for every hole X .

Starting from the full semantics of small networks and holes, we define by induction the full semantics $\llbracket \mathcal{N} \rrbracket$ of a network specification \mathcal{N} in general. In a similar way and simultaneously, 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 prefer to define $\llbracket \mathcal{N} \rrbracket$ separately first, and then define $\langle\langle \mathcal{N} \rangle\rangle$ from $\llbracket \mathcal{N} \rrbracket$. 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:

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

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

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

where $[f]_{\mathbf{in}(\mathcal{M})}$, $[f]_{\mathbf{out}(\mathcal{M})}$, and $[f]_{\#(\mathcal{M})}$, are the restrictions of the function 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 let $g \in \llbracket \mathcal{N} \rrbracket$. We define $f \parallel g$ as follows:

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

The operation “ \parallel ” on flows is associative, but not commutative, just as the related constructor “ \parallel ” on network specifications.

We define the full semantics $\llbracket \mathcal{M} \rrbracket$ 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 $\llbracket \mathcal{M} \rrbracket = \llbracket \mathcal{A} \rrbracket$.

2. If $\mathcal{M} = {}^iX$, then $\llbracket \mathcal{M} \rrbracket = {}^i\llbracket X \rrbracket$.
3. If $\mathcal{M} = (\mathcal{P}_1 \parallel \mathcal{P}_2)$, then $\llbracket \mathcal{M} \rrbracket = \{ (f_1 \parallel f_2) \mid f_1 \in \llbracket \mathcal{P}_1 \rrbracket \text{ and } f_2 \in \llbracket \mathcal{P}_2 \rrbracket \}$.
4. If $\mathcal{M} = (\mathbf{let } X = \mathcal{P} \mathbf{ in } \mathcal{P}')$, then $\llbracket \mathcal{M} \rrbracket = \llbracket \mathcal{P}' \rrbracket$, provided two conditions:
 - (a) $\dim(X) \approx \dim(\mathcal{P})$,
 - (b) for every $f : \mathbf{in}(X) \cup \mathbf{out}(X) \rightarrow \mathbb{R}^+$,

$$f \in \llbracket X \rrbracket \quad \text{iff} \quad \text{there is } g \in \llbracket \mathcal{P} \rrbracket \text{ such that } f \approx [g]_A$$

where $A = \mathbf{in}(\mathcal{P}) \cup \mathbf{out}(\mathcal{P})$.

5. If $\mathcal{M} = \mathbf{bind}(\mathcal{P}, \langle a, b \rangle)$, then $\llbracket \mathcal{M} \rrbracket = \{ f \mid f \in \llbracket \mathcal{P} \rrbracket \text{ and } f(a) = f(b) \}$.

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

$$\langle\langle \mathcal{N} \rangle\rangle = \{ [f]_A \mid f \in \llbracket \mathcal{N} \rrbracket \}$$

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

Remark 14. 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} , $L_{\mathcal{N}}$ 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 also 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 $\mathcal{M} = \mathbf{bind}(\mathcal{P}, \langle a, b \rangle)$, 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. \square

Remark 15. We do not disallow the possibility that $\llbracket \mathcal{N} \rrbracket = \emptyset$, which happens when there are no feasible flows in \mathcal{N} . For example, if \mathcal{N} mentions only one small network \mathcal{A} and there are no feasible flows in \mathcal{A} , then it must be that $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{N} \rrbracket = \emptyset$, which also implies $\langle\langle \mathcal{A} \rangle\rangle = \emptyset$.

Another possibility is $\langle\langle \mathcal{N} \rangle\rangle = \{\emptyset\}$, different from the preceding, the result of inductively defining a *totally closed* \mathcal{N} , i.e., \mathcal{N} has no input arcs and no output arcs. In such a case, there are *internal* feasible flows only.

There are other special cases, when $\mathbf{in}(\mathcal{N}) = \emptyset$ or $\mathbf{out}(\mathcal{N}) = \emptyset$, but not both. For example, if $\mathbf{in}(\mathcal{N}) = \{a_1, \dots, a_k\}$ for some $k \geq 1$ and $\mathbf{out}(\mathcal{N}) = \emptyset$, then either $\langle\langle \mathcal{N} \rangle\rangle = \emptyset$ (there are no feasible flows in \mathcal{N}) or $\langle\langle \mathcal{N} \rangle\rangle = \{f\}$ with $f(a_1) = \dots = f(a_k) = 0$. \square

Remark 16. It is possible to define rewriting (or reduction) rules on closed network specifications in order to reduce each into an equivalent finite set of closed network specifications in *normal form*, a normal form being free of let-bindings. We can do this in such a way that the semantics, as just defined, are an *invariant* of the transformation, which will thus be *sound* relative to our chosen semantics. But this comes with a price of dealing with several technical complications, as already alluded to in Remarks 9 and 12. \square

Remark 17. Intermediate forms, between network specifications in general and their *normal forms*, are what we may call *canonical forms*. We define the latter in a more structured way so that they will always appear as:

$$\begin{aligned} & \mathbf{let} \ X_1 \in \{\mathcal{M}_{1,1}, \dots, \mathcal{M}_{1,k_1}\} \ \mathbf{in} \\ & \mathbf{let} \ X_2 \in \{\mathcal{M}_{2,1}, \dots, \mathcal{M}_{2,k_2}\} \ \mathbf{in} \\ & \quad \vdots \\ & \mathbf{let} \ X_\ell \in \{\mathcal{M}_{\ell,1}, \dots, \mathcal{M}_{\ell,k_\ell}\} \ \mathbf{in} \ \mathbf{bind}(\mathcal{P}_1 \parallel \mathcal{P}_2 \parallel \dots \parallel \mathcal{P}_m, \theta) \end{aligned}$$

where $\mathcal{M}_{i,j}$ is in canonical form, for every $1 \leq i \leq \ell$ and $1 \leq j \leq k_i$, and every member of $\{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m\}$ is a small network or a hole. As used here **let** and **bind** are the derived constructors defined in Section 3.1. We can show that every network specification \mathcal{N} can be uniquely transformed, via rewriting rules omitted here, into such a canonical form \mathcal{N}' without incurring an exponential explosion in size, as will generally happen when transforming into normal form. \mathcal{N}' is essentially obtained from \mathcal{N} by re-arranging the order in which constructors are used. Moreover, we can do the transformation so that $\llbracket \mathcal{N} \rrbracket = \llbracket \mathcal{N}' \rrbracket$, thus proving its soundness.

One benefit of canonical forms is to facilitate some of the proofs by structural inductions on \mathcal{N} , such as that of Proposition 18. If \mathcal{N} is in canonical form and \mathcal{N} is closed, then the induction can be directed so that the full semantics $\llbracket X \rrbracket$ of a hole X does not need to be “guessed” when we define $\llbracket \mathbf{let} \ X = \mathcal{P} \ \mathbf{in} \ \mathcal{P}' \rrbracket$ from $\llbracket \mathcal{P} \rrbracket$ and $\llbracket \mathcal{P}' \rrbracket$: We first determine $\llbracket \mathcal{P} \rrbracket$, then directly define $\llbracket X \rrbracket$ from $\llbracket \mathcal{P} \rrbracket$ by restricting members of the latter to $\mathbf{in}(\mathcal{P}) \cup \mathbf{out}(\mathcal{P})$, and then substitute renamed copies of $\llbracket X \rrbracket$ in the occurrences of X in \mathcal{P}' , before proceeding to determine $\llbracket \mathcal{P}' \rrbracket$ without having to consider $\llbracket X \rrbracket$ as a base case in the induction. \square

The following proposition establishes a crucial link between the IO-semantics of a network specification \mathcal{N} and its possible typings, as introduced in later sections.

Proposition 18 (IO-Semantics and Types). *Let \mathcal{N} be a closed network specification, with $\mathbf{A}_{\mathbf{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\mathbf{out}} = \mathbf{out}(\mathcal{N})$. For every $\emptyset \neq A \subseteq \mathbf{A}_{\mathbf{in}} \cup \mathbf{A}_{\mathbf{out}}$, define the two quantities:*

$$\begin{aligned} s_{\min}(A) &= \min \left\{ \sum f(A \cap \mathbf{A}_{\mathbf{in}}) - \sum f(A \cap \mathbf{A}_{\mathbf{out}}) \mid f \in \llbracket \mathcal{N} \rrbracket \right\} \\ s_{\max}(A) &= \max \left\{ \sum f(A \cap \mathbf{A}_{\mathbf{in}}) - \sum f(A \cap \mathbf{A}_{\mathbf{out}}) \mid f \in \llbracket \mathcal{N} \rrbracket \right\} \end{aligned}$$

For every $t \in \{s_{\min}(A), \dots, s_{\max}(A)\}$, there is $f \in \llbracket \mathcal{N} \rrbracket$ such that $t = \sum f(A \cap \mathbf{A}_{\mathbf{in}}) - \sum f(A \cap \mathbf{A}_{\mathbf{out}})$.

Later, $[s_{\min}(A), s_{\max}(A)]$ will be the type/interval assigned by a principal typing of \mathcal{N} to a set A of input and output arcs. In words, the proposition asserts that every t in the interval assigned to A is assumed by some feasible flow in \mathcal{N} , and that no t outside this interval is assumed by any feasible flow. Thus, the interval assigned to A exactly includes all the values witnessed by feasible flows in \mathcal{N} and no other values.

Proof Sketch. By the definition of $\llbracket \mathcal{N} \rrbracket$ from $\llbracket \mathcal{N} \rrbracket$, we can replace “ $\llbracket \mathcal{N} \rrbracket$ ” by “ $\llbracket \mathcal{N} \rrbracket$ ” throughout the statement of the proposition. The proof is by induction on the definition $\llbracket \mathcal{N} \rrbracket$. To facilitate the induction, we put \mathcal{N} in canonical form, according to Remark 17, so that the base case in the induction is limited to $\llbracket \mathcal{A} \rrbracket$ for all the small networks \mathcal{A} in \mathcal{N} , with no need to consider $\llbracket X \rrbracket$ for the holes X occurring in \mathcal{N} . \square

4.1 Flow Conservation, Capacity Constraints, Type Satisfaction (Continued)

The fundamental concepts stated in relation to small networks \mathcal{A} in Section 2.1 can be now extended to arbitrary network specifications \mathcal{N} .

Definition 19 (Flow Conservation – Continued). All the nodes mentioned by \mathcal{N} are all the nodes in the small networks occurring in \mathcal{N} , because our inductive definition in Section 3 does not introduce new nodes. We can therefore say that the full semantics $\llbracket \mathcal{N} \rrbracket$ *satisfies flow conservation* because, for every small network \mathcal{A} in \mathcal{N} , every $f \in \llbracket \mathcal{A} \rrbracket$ satisfies flow conservation at every node, *i.e.*, the equation in (1) in Definition 4. \square

Definition 20 (*Capacity Constraints – Continued*). The only arcs introduced by our inductive definition in Section 3, 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 precisely set in order not to conflict with those already defined on the input/output arcs of small networks, as explained in Remark 14.

We can therefore say that the semantics $\llbracket \mathcal{N} \rrbracket$ *satisfies the capacity constraints* because, for every small network \mathcal{A} appearing in \mathcal{N} , every flow $f \in \llbracket \mathcal{A} \rrbracket$ satisfies the capacity constraints on every arc, *i.e.*, the inequalities in (2) in Definition 5. \square

Definition 21 (*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

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

which may, or may not, be satisfied by $f \in \langle\langle \mathcal{N} \rangle\rangle$ or by $f \in \llbracket \mathcal{N} \rrbracket$. We say $f \in \langle\langle \mathcal{N} \rangle\rangle$ or $f \in \llbracket \mathcal{N} \rrbracket$ *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:

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

The inequalities in (4) are the same as in (3) in Definition 7, now extended to network specifications in general.

One special case not covered by (4) is when $\llbracket \mathcal{N} \rrbracket = \emptyset$, *i.e.*, there are no feasible flows in \mathcal{N} , in which case also $\langle\langle \mathcal{N} \rangle\rangle = \emptyset$. An appropriate typing T for \mathcal{N} in this case assigns the empty interval to every $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$.

We may restrict attention to the input arcs or to the output arcs, in which case we say $f \in \langle\langle \mathcal{N} \rangle\rangle$ or $f \in \llbracket \mathcal{N} \rrbracket$ *satisfies* the typing T *at the input* or *at the output*, respectively. If the restriction is to the input, then for every $A \subseteq \mathbf{A}_{\text{in}}$ with $[T]_{\text{in}}(A) = [r, r']$, we have:

$$(5) \quad r \leq \sum f(A) \leq r'$$

or if it is to the output, then for every $B \subseteq \mathbf{A}_{\text{out}}$ with $[T]_{\text{out}}(B) = [s, s']$, we have:

$$(6) \quad s \leq -\sum f(B) \leq s'$$

If $f \in \langle\langle \mathcal{N} \rangle\rangle$ or $f \in \llbracket \mathcal{N} \rrbracket$ satisfies T at the input, we may say f *satisfies* $[T]_{\text{in}}$, and if at the output, we say f *satisfies* $[T]_{\text{out}}$. \square

It is worth stressing that, while satisfaction of (4) implies satisfaction of both (5) and (6), the converse is not necessarily true: It may happen that f satisfies (5) and (6) but not (4). Example 40 below illustrates this point.

Remark 22. If there is a set K of several commodities, then (4) must involve summations over K , as follows:

$$r \leq \sum_{\kappa \in K} \sum f_{\kappa}(A \cap \mathbf{A}_{\text{in}}) - \sum_{\kappa \in K} \sum f_{\kappa}(A \cap \mathbf{A}_{\text{out}}) \leq r'$$

where f_{κ} is the flow of commodity κ . As mentioned already (Remark 2), nothing essential is lost by restricting attention to one commodity. \square

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 $m \geq 0$. As usual, we assume there is a fixed ordering on the arcs in \mathbf{A}_{in} and again on the arcs in \mathbf{A}_{out} . With no loss of generality, suppose:

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

where $\ell \leq m$. 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. This notational convention is used repeatedly in later sections.

This notation is a useful reminder that any flow through the arcs $\{a_1, \dots, a_k\}$ contributes a *positive* quantity, and that through the arcs $\{a_{k+1}, \dots, a_\ell\}$ a *negative* quantity, and that these two quantities together should add up to a value within the interval $[r, r']$.

There is still another advantage to this notation, as it provides a more direct connection with aspects of our later analysis based on linear spaces. A typing T for $\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$ induces a *polytope* (or *bounded polyhedron*), which we call $\text{Poly}(T)$, in the Euclidean hyperspace \mathbb{R}^m , as we explain next.

We think of the m arcs in $\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$ as referring to the m dimensions of the space \mathbb{R}^m . $\text{Poly}(T)$ is the non-empty intersection of at most $2 \cdot (2^m - 1)$ halfspaces. Indeed, there are $(2^m - 1)$ non-empty subsets in $\mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$, and the bounded interval $[r, r']$ which T assigns to such a subset $A = \{a_1, \dots, a_\ell\}$, as above, induces two linear inequalities in the variables $\{a_1, \dots, a_\ell\}$, denoted $T_{\geq}(A)$ and $T_{\leq}(A)$, namely:

$$(7) \quad \begin{aligned} T_{\geq}(A) : \quad & a_1 + \cdots + a_k - a_{k+1} - \cdots - a_\ell \geq r \\ T_{\leq}(A) : \quad & a_1 + \cdots + a_k - a_{k+1} - \cdots - a_\ell \leq r' \end{aligned}$$

and, therefore, two halfspaces $\text{Half}(T_{\geq}(A))$ and $\text{Half}(T_{\leq}(A))$:

$$(8) \quad \begin{aligned} \text{Half}(T_{\geq}(A)) &= \{ \mathbf{r} \in \mathbb{R}^m \mid \mathbf{r} \text{ satisfies inequality } T_{\geq}(A) \} \\ \text{Half}(T_{\leq}(A)) &= \{ \mathbf{r} \in \mathbb{R}^m \mid \mathbf{r} \text{ satisfies inequality } T_{\leq}(A) \} \end{aligned}$$

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

$$\text{Poly}(T) = \bigcap \{ \text{Half}(T_{\geq}(A)) \cap \text{Half}(T_{\leq}(A)) \mid \emptyset \neq A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \}$$

Generally, many of the inequalities induced by the typing T will be redundant, and the induced $\text{Poly}(T)$ 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 $\text{Poly}(T)$ being a polytope in Euclidean hyperspace. As in other situations connecting syntax and semantics, there are generally distinct typings T and T' such that $\text{Poly}(T) = \text{Poly}(T')$. 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 later want to 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:

$$(9) \quad \{ T_{\geq}(A) \mid \emptyset \neq A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \} \cup \{ T_{\leq}(A) \mid \emptyset \neq A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \}$$

where $T_{\geq}(A)$ and $T_{\leq}(A)$ are as in (7) 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. However, even if we agree on a canonical order in which to carry out the elimination of redundant inequalities, the same inequality retained at the end can be written in different forms, e.g., $2a_1 - (1/2)a_2 \leq 4$ is equivalent to $a_1 - (1/4)a_2 \leq 2$ and $4a_1 - a_2 \leq 8$ and many others. We therefore need to apply some care when using such elimination procedures, if we want to uniquely produce non-redundant typings. Some of these issues are illustrated in the next example.

Example 23. Consider the small network \mathbf{M} from Example 13, where we now assign capacities to the arcs, as shown in Figure 3. The number in rectangular boxes are upper-bounds capacities; all other capacity bounds, not appearing in the figure, are trivial, *i.e.*, the lower bound for all arcs in $\{d_1, d_2, d_3\}$ is 0.

For this example, $\mathbf{A}_{\text{in}} = \{d_1, d_2\}$ and $\mathbf{A}_{\text{out}} = \{d_3\}$. A typing T assigns an interval to each of the 7 non-empty subsets in $\mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$, which we choose here as:

$$\begin{array}{lll} (i) & d_1 : [0, 10] & (ii) \quad d_2 : [0, 20] & (iii) \quad -d_3 : [-20, 0] \\ (iv) & d_1 + d_2 : [0, 20] & (v) \quad d_1 - d_3 : [-20, 0] & (vi) \quad d_2 - d_3 : [-10, 0] \\ (vii) & d_1 + d_2 - d_3 : [0, 0] & & \end{array}$$

which in turn correspond to 14 inequalities. Here, 4 of the 7 interval assignments are redundant and can be eliminated, but the elimination is not unique. The simplest perhaps is to eliminate $\{(ii), (iv), (v), (vi)\}$, and to keep $\{(i), (iii), (vii)\}$ which are:

$$(i) \quad d_1 : [0, 10] \quad (iii) \quad -d_3 : [-20, 0] \quad (vii) \quad d_1 + d_2 - d_3 : [0, 0]$$

Call T_1 the resulting *partial* interval-assignment (formally introduced in Definition 37). An alternative is to keep only $\{(i), (iv), (vii)\}$ and eliminate the other intervals, resulting in another partial typing T_2 . Both T_1 and T_2 are equivalent to T , because $\text{Poly}(T) = \text{Poly}(T_1) = \text{Poly}(T_2)$ in \mathbb{R}^3 . There are still other partial typings equivalent to T .

In the terminology of Section 5.2 below, the particular typing T here is *valid* but not *principal*. That T is not principal is not a consequence of the redundant inequalities it induces, but of the intervals it assigns being narrower than necessary: We obtain a principal typing \tilde{T} by widening some of the intervals T assigns, specifically, by changing “10” to “15” and “20” to “35” throughout. If we make the same changes in T_1 and T_2 , we obtain partial typings \tilde{T}_1 and \tilde{T}_2 equivalent to \tilde{T} . \square

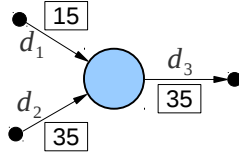


Figure 3: An assignment of capacities to the arcs of small network \mathbf{M} in Examples 13 and 23.

Definition 24 (Projections and Restrictions). Let $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$ as in the opening paragraph of Section 5. If $\mathbf{r} = \langle r_1, \dots, r_m \rangle$ is an arbitrary point in \mathbb{R}^m , then the *projection* of \mathbf{r} on the ℓ -dimensional subspace defined by A , written $\text{Proj}_A(\mathbf{r})$, is obtained by omitting all the entries in \mathbf{r} corresponding to the coordinates *not* in A , *i.e.*, the coordinates in $(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) - \{a_1, \dots, a_\ell\}$, so that:

$$\text{Proj}_A(\mathbf{r}) = \langle r_1, \dots, r_\ell \rangle$$

Consider a typing $T : \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$. The *restriction of T to A* is defined by:

$$(10) \quad [T]_A(B) = \begin{cases} T(B) & \text{if } B \subseteq A, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Our earlier notations $[T]_{\text{in}}$ and $[T]_{\text{out}}$ denote the functions $[T]_{\mathbf{A}_{\text{in}}}$ and $[T]_{\mathbf{A}_{\text{out}}}$ here. $[T]_{\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}}$ is exactly T . We write $[T]_a$ instead of $[T]_{\{a\}}$ for a single arc $a \in \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$. \square

Definition 25 (*Tight Typings*). Typing T is *tight* for $A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$, if it is the case that for every $B \subseteq A$:

$$(11) \quad \text{Proj}_B(\text{Poly}(T)) = \text{Poly}([T]_B)$$

Informally, if we view the restriction $[T]_A$ as a projection on A , then the preceding equality says “the projection of the polytope is equal to the polytope of the projection”.

We say that T is *uniformly tight* if it is tight for every $A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$. A uniformly tight typing discards redundancies different from those considered in Example 23, where we eliminated linearly dependent inequalities by standard procedures of linear algebra. Example 30 illustrates the kind of redundancies excluded by uniformly tight typings. \square

Proposition 26 (First Orthant Contains $\text{Poly}(T)$). *Let $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$ as in the opening paragraph of Section 5. Let $T : \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$ be a typing for \mathcal{N} . If $\mathbf{r} = \langle r_1, \dots, r_m \rangle \in \text{Poly}(T)$, then all the coordinates r_1, \dots, r_m are non-negative, i.e., \mathbf{r} is entirely located in the first orthant of the m -dimensional hyperspace \mathbb{R}^m . (Informally, this makes sense, because flow on every input/output arc must be a non-negative value.)*

Proof. Suppose T is tight for all the singleton subsets of $\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$. Consider the intervals assigned by T to all these singleton subsets. There are m such intervals $[r_1, r'_1], [r_2, r'_2], \dots, [r_m, r'_m]$. If $a_i \in \mathbf{A}_{\text{in}}$, then its type $[r_i, r'_i]$ is such that $0 \leq r_i \leq r'_i$, so that the induced inequalities $T_{\geq}(a_i)$ and $T_{\leq}(a_i)$ are:

$$0 \leq r_i \leq a_i \leq r'_i$$

which force all values assigned to input arc a_i to be non-negative. If $a_j \in \mathbf{A}_{\text{out}}$, then its type $[r_j, r'_j]$ is such that $r_j \leq r'_j \leq 0$, so that the induced inequalities $T_{\geq}(a_j)$ and $T_{\leq}(a_j)$ are:

$$\leq r_j \leq -a_j \leq r'_j \leq 0$$

which force all values assigned to output arc a_j to be non-negative. Hence, these m intervals define an axis-aligned hyperrectangle enclosing $\text{Poly}(T)$ entirely within the first orthant of the hyperspace \mathbb{R}^m . \square

Proposition 27 (Every Typing Is Equivalent to a Uniformly Tight Typing). *There is an algorithm $\text{Tight}()$ such that, given an arbitrary typing $(\mathcal{N} : T)$ as input, will always terminate and return an equivalent uniformly tight typing $(\mathcal{N} : \text{Tight}(T))$, i.e., such that $\text{Poly}(T) = \text{Poly}(\text{Tight}(T))$ and $\text{Tight}(T)$ is uniformly tight.*³

Proof Sketch. Let $|\dim(\mathcal{N})| = m$. According to Proposition 26, $\text{Poly}(T)$ is entirely contained within the first orthant of the m -dimensional hyperspace. There are $2 \cdot (2^m - 1)$ induced inequalities of the form $T_{\geq}(A)$ and $T_{\leq}(A)$, where $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$ with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$, according to (7) earlier in this section. We can implement the desired algorithm $\text{Tight}()$ as a repeated application of a linear programming algorithm, twice to every nonempty $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, in order to compute the minimum r and the maximum r' of the following linear expression E :

$$E = \left(\sum A \cap \mathbf{A}_{\text{in}} \right) - \left(\sum A \cap \mathbf{A}_{\text{out}} \right)$$

where we use the arcs in A as variables, over the polytope $\text{Poly}(T)$. The interval $[r, r']$ is precisely the interval that a uniformly tight typing must assign to A . \square

Corollary 28. *$(\mathcal{N} : T)$ is a uniformly tight typing iff $T = \text{Tight}(T)$.*

Proposition 29 (Projection of Polytope Contained in Polytope of Projection). *Let $(\mathcal{N} : T)$ be a typing, with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. The following assertions hold, for every $\emptyset \neq A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$:*

³For now, we leave out the issue of the efficiency of algorithm $\text{Tight}()$.

1. $\text{Proj}_A(\text{Poly}(T)) \subseteq \text{Poly}([T]_A)$.
2. If T is tight for A and $f_0 : A \rightarrow \mathbb{R}^+$ satisfies $[T]_A$, then f_0 can be extended to $f : \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \rightarrow \mathbb{R}^+$ which satisfies T .

Proof Sketch. For both parts of the proposition, suppose $|\dim(\mathcal{N})| = m \geq 1$ and, with no loss of generality, let $A = \{a_1, \dots, a_\ell\}$ for some $1 \leq \ell \leq m$.

For part 1, consider an arbitrary $\mathbf{r} = \langle r_1, \dots, r_m \rangle \in \text{Poly}(T)$. Define $\mathbf{r}_0 = \text{Proj}_A(\mathbf{r}) = \langle r_1, \dots, r_\ell \rangle$. We have to show that $\mathbf{r}_0 \in \text{Poly}([T]_A)$. By the definition of $[T]_A$ in (10), this is equivalent to showing that \mathbf{r}_0 satisfies the two induced inequalities in (7) for every $B \subseteq A$. This last assertion is a straightforward consequence of the definitions.

For part 2, let f_0 be represented by $\mathbf{r}_0 = \langle r_1, \dots, r_\ell \rangle$. If f_0 satisfies $[T]_A$, then \mathbf{r}_0 is a point in $\text{Poly}([T]_A)$. Because T is tight for A , this implies \mathbf{r}_0 is a point in $\text{Proj}_A(\text{Poly}(T))$. This means there is a point $\mathbf{r} \in \text{Poly}(T)$ such that $\mathbf{r}_0 = \text{Proj}_A(\mathbf{r})$, which in turn implies the desired conclusion. \square

Example 30. Consider again the typing T defined in Example 23. It is a straightforward exercise to check that T satisfies (11) for every $A \subseteq \{d_1, d_2, d_3\}$, implying that T is uniformly tight.

Define a new typing T' from T by making a single change in it, namely, change the interval assignment for d_2 from “ $d_2 : [0, 20]$ ” to “ $d_2 : [0, 30]$ ”.

It is easy to see that this change is without effect on the meaning of the typing, *i.e.*, $\text{Poly}(T) = \text{Poly}(T')$, so that if we project on $A = \{d_2\}$ we obtain the equality:

$$\text{Proj}_{d_2}(\text{Poly}(T)) = \text{Proj}_{d_2}(\text{Poly}(T')) = \{r \in \mathbb{R} \mid 0 \leq r \leq 20\}$$

However, $[T]_{d_2} = [0, 20]$ while $[T']_{d_2} = [0, 30]$, which implies;

$$\{r \in \mathbb{R} \mid 0 \leq r \leq 20\} = \text{Poly}([T]_{d_2}) \neq \text{Poly}([T']_{d_2}) = \{r \in \mathbb{R} \mid 0 \leq r \leq 30\}$$

Hence, T' is not tight for $\{d_2\}$ and, *a fortiori*, not uniformly tight for every set of arcs containing d_2 . A graphical explanation limited to $\{d_1, d_2\}$ is given in Figure 4. \square

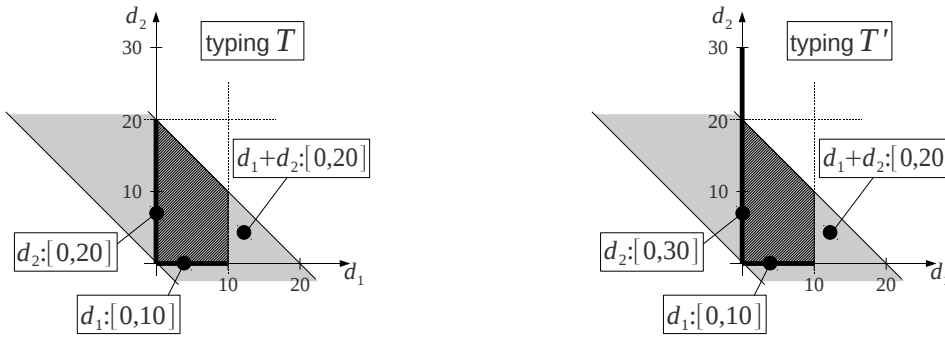


Figure 4: Graphic explanation for Example 30. T and T' are equivalent typings, T is uniformly tight, T' is not. T is tight for $\{d_1, d_2\}$, $\{d_1\}$ and $\{d_2\}$, whereas T' is tight for $\{d_1\}$ but not for $\{d_1, d_2\}$ and $\{d_2\}$.

Let \mathcal{N} be a network, with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. In this report, we use typings as total mappings from $\mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$ to $\mathbb{R} \times \mathbb{R}$, leaving for future work the question of how to uniquely and minimally represent typings by equivalent *partial* typings (see Open Problem 38).

As for the other kind of redundancy, resulting from non-tight typings, we use algorithm $\text{Tight}()$ defined in Proposition 27 whenever needed, and we leave for future work the question of improving $\text{Tight}()$'s efficiency.

5.2 Valid Typings and Principal Typings

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

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

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 $f \in \llbracket \mathcal{N} \rrbracket$ satisfies T .

More succinctly, $\mathcal{N} : T$ is *valid* iff $\text{Poly}(T) \subseteq \llbracket \mathcal{N} \rrbracket$, and $\mathcal{N} : T$ is *principal* iff $\text{Poly}(T) = \llbracket \mathcal{N} \rrbracket$.

If there are no feasible flows in \mathcal{N} , then the empty typing $T = \emptyset$, *i.e.*, the typing that assigns the empty interval to every $A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$, is a principal (and only valid) typing for \mathcal{N} . No feasible flow satisfies \emptyset . In this case, $\text{Poly}(T) = \emptyset$.

If $\mathcal{N}_1 : T_1$ and $\mathcal{N}_2 : T_2$ are typings for networks \mathcal{N}_1 and \mathcal{N}_2 with similar input and output dimensions, we write $T_1 \equiv T_2$ whenever $\text{Poly}(T_1) \approx \text{Poly}(T_2)$ and say that T_1 and T_2 are *equivalent*.

A common and useful notion in type theories is *subtyping*. If T_1 is a *subtype* of T_2 , which we write $T_1 <: T_2$ to follow convention, 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) $T_1 <: T_2$ iff $\text{Poly}(T_2) \subseteq \text{Poly}(T_1)$.

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 31 (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 $T_1 <: T_2$.*

Proof. Given an arbitrary $f : \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \rightarrow \mathbb{R}^+$, we want to show that if f satisfies T_2 , then f satisfies T_1 , *i.e.*, any point in $\text{Poly}(T_2)$ is also in $\text{Poly}(T_1)$. If f satisfies T_2 , then f can be extended to a feasible flow f' . Because T_1 is principal, f' satisfies T_1 . This implies that the restriction of f' to $\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, which is exactly f , satisfies T_1 . \square

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 Proposition 33. That this is not the case for valid typings is illustrated by Example 23, where T and \tilde{T} are both valid but $\text{Poly}(T) \neq \text{Poly}(\tilde{T})$.

Lemma 32. *Let $(\mathcal{N} : T)$ and $(\mathcal{N} : T')$ be typings for the same \mathcal{N} . If T and T' are uniformly tight and $\text{Poly}(T) = \text{Poly}(T')$, then $T = T'$.*

Proof. This is a straightforward consequence of Proposition 27 and its Corollary 28. \square

Proposition 33 (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 uniformly tight, then $T_1 = T_2$.*

Proof. Both $(\mathcal{N} : T_1)$ and $(\mathcal{N} : T_2)$ are valid. Hence, by Proposition 31, both $T_1 <: T_2$ and $T_2 <: T_1$. This implies that $T_1 \equiv T_2$. When T_1 and T_2 are uniformly tight, then the equality $T_1 = T_2$ follows from Lemma 32. \square

Corollary 34. *Let \mathcal{N} be a network specification. Among the valid typings for \mathcal{N} , the principal typings are all equivalent and minimal w.r.t. to the subtyping ordering “<:”.*

Proof. Immediate from Propositions 31 and 33. \square

6 Other Properties and Open Problems of Network Typings

Let \mathcal{N} be a network specification, and let $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. All valid typings for \mathcal{N} share symmetries and other properties that make their manipulation easier and more efficient.

We define two operations, “+” and “−”, on non-empty types/intervals. Given two intervals of reals, $[r_1, r'_1]$ and $[r_2, r'_2]$ where $r_1 \leq r'_1$ and $r_2 \leq r'_2$, we define:

$$[r_1, r'_1] + [r_2, r'_2] = [r_1 + r_2, r'_1 + r'_2]$$

and for an interval of non-negative reals $[r, r']$, where $0 \leq r \leq r'$, we define:

$$-[r, r'] = [-r', -r]$$

so that $-[r, r']$ is an interval of non-positive reals. If T is a valid network typing other than \emptyset , then T will have the following properties, which we state without proof:

1. If T is tight for $\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, then:

- (a) $T(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) = [0, 0]$.
- (b) $T(\mathbf{A}_{\text{in}}) \subseteq \mathbb{R}^+$.
- (c) $T(\mathbf{A}_{\text{in}}) = -T(\mathbf{A}_{\text{out}})$.

2. For all $A, B \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, we have $T(A \cup B) \subseteq T(A) + T(B)$.

3. For all $A, B \subseteq \mathbf{A}_{\text{in}}$, we have $T(A \cap B) \subseteq T(A) \cap T(B)$.

4. For all $A, B \subseteq \mathbf{A}_{\text{out}}$, we have $T(A \cap B) \subseteq T(A) \cap T(B)$.

Open Problem 35. Let \mathbf{A}_{in} and \mathbf{A}_{out} be fixed sets of input and output arcs, and let \mathcal{N} range over all network specifications such that $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. For all such \mathcal{N} , the dimensions are the same and fixed, $|\dim(\mathcal{N})| = |\dim_{\text{in}}(\mathcal{N}) \cdot \dim_{\text{out}}(\mathcal{N})| = m \geq 1$.

Let T range over functions from $\mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$ to $\mathbb{R} \times \mathbb{R}$. Not every such function is a principal typing for some \mathcal{N} . We want two characterizations of the principal typings:

(syntactical) Consider the intervals $[r, r']$ assigned by T and the way they appear in the two inequalities induced by T – namely, $T_{\geq}(A)$ and $T_{\leq}(A)$ in (7), for every $A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$. State and prove necessary and sufficient conditions on these intervals so that: T is uniformly tight *and* T is a principal typing for \mathcal{N} .

(semantical) Consider polytopes P in the first orthant of the m -dimensional hyperspace \mathbb{R}^m . State and prove necessary and sufficient conditions so that $P = \text{Poly}(T)$ for a typing T which is principal for \mathcal{N} .

When $|\dim_{\text{in}}(\mathcal{N})| = |\dim_{\text{out}}(\mathcal{N})| = 1$, the two preceding characterizations are trivial. When $|\dim_{\text{in}}(\mathcal{N})| = 1$ or $|\dim_{\text{out}}(\mathcal{N})| = 1$, but not both, they still appear relatively easy to formulate and prove. The situation is more complicated when both $|\dim_{\text{in}}(\mathcal{N})| \geq 2$ and $|\dim_{\text{out}}(\mathcal{N})| \geq 2$, as illustrated by the conjecture to follow. \square

Conjecture 36. We state a claim, below, and conjecture that it is true. We need a few preliminary definitions for a precise statement.

Consider all small network \mathcal{A} , with the same two input arcs and the same two output arcs, say, $\mathbf{in}(\mathcal{A}) = \{a_1, a_2\}$ and $\mathbf{out}(\mathcal{A}) = \{a_3, a_4\}$.

Set the lower-bound capacities to zero on all arcs; in particular, $L(a_1) = L(a_2) = L(a_3) = L(a_4) = 0$. We specify the upper-bound capacities on the input arcs and output arcs only: $U(a_1) = U(a_3) = 15$ and $U(a_2) = U(a_4) = 25$, leaving the upper-bound capacities on the internal arcs unspecified.

Claim: If a typing T for \mathcal{A} includes the following type assignments, then T is not principal (though it may be valid) for \mathcal{A} :

$$\begin{aligned}
\text{(a)} \quad T(\{a_1\}) &= [0, 15], & T(\{a_2\}) &= [0, 25], & T(\{a_1, a_2\}) &= [0, 30] \\
\text{(b)} \quad T(\{a_3\}) &= -[0, 15], & T(\{a_4\}) &= -[0, 25], & T(\{a_3, a_4\}) &= -[0, 30] \\
\text{(c)} \quad T(\{a_1, a_3\}) &= [-5, 5], & T(\{a_2, a_4\}) &= [-5, 5].
\end{aligned}$$

Line (a) and line (b) specify the types at the input and output, respectively. The unspecified internal structure of \mathcal{A} , and the unspecified upper-bound capacities on internal arcs, must be such that the maximum flow across a “minimum cut” is 30, which is the upper end in the type assignment $T(\{a_1, a_2\}) = -T(\{a_3, a_4\}) = [0, 30]$. Note that the maximum flow that input arcs a_1 and a_2 (or output arcs a_3 and a_4) can carry is $15 + 25 = 40$, which exceeds the max flow 30 of a “minimum cut”.

There are small networks \mathcal{A} whose principal typings include the type assignments in lines (a) and (b), but not in line (c). Such small networks are the two in Example 40. The complication arises from the type assignments in line (c). However, the same two small networks have valid typings that include all the type assignments in lines (a), (b), and (c), as shown in Example 41 below.

If our claim can be verified, it will show that *not* every function T from $\mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$ to $\mathbb{R} \times \mathbb{R}$ is a principal network typing, even if T satisfies all the restrictions spelled out earlier in Section 5. \square

Definition 37 (Partial Typings). Let \mathcal{N} be a network specification, with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. A *partial typing* T for \mathcal{N} is a partial function:

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

Such a partial mapping can be extended to a total mapping $T' : \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$, in which case we may 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 T is *principal* for \mathcal{N} if *every* (total) typing $T' \supseteq T$ is principal for \mathcal{N} .

Let $|\dim(\mathcal{N})| = m \geq 1$. We say the partial typing T for \mathcal{N} is *minimal* if for every partial typing T' for \mathcal{N} such that $T' \subsetneq T$, *i.e.*, for every T' which assigns fewer intervals than T , it is the case that $T \not\subseteq T'$. Note that if $T \not\subseteq T'$ and $T' \subseteq T$, not $T \subseteq T'$, then the polyhedron of T is strictly contained in that of T' , *i.e.*, $\text{Poly}(T) \subsetneq \text{Poly}(T')$, in the hyperspace \mathbb{R}^m .

We say the partial typing T is *uniformly tight* if for every partial typing T' such that $T \equiv T'$, it is the case that for every $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, if both $T(A)$ and $T'(A)$ are defined, then the interval $T(A)$ is contained in the interval $T'(A)$, *i.e.*, $T(A) \subseteq T'(A)$.

(The “syntactic” definition of uniformly tight typings here coincide with the “semantic” one in Definition 25 when typings are total. A total typing is a particular case of a partial typing.) \square

Open Problem 38. Let \mathcal{N} be a network, with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$. In general, valid and principal typings for \mathcal{N} are “over-specified”, as they unnecessarily assign intervals to all the subsets in $\mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$. Over-specified typings are not minimal and wasteful of computational resources. The typings T and \tilde{T} for the small network \mathbf{M} in Example 23 are not minimal, whereas the partial valid T_1 and T_2 , and the partial principal \tilde{T}_1 and \tilde{T}_2 , are minimal.

We want an algorithm $\text{Canonical}()$ which, given a valid (or, in particular, principal) typing T for \mathcal{N} as a function $T : \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$, be it total or partial, will *uniquely* determine an equivalent partial typing T' which is *both* minimal and uniformly tight. As an extra, examine the (lower bound) complexity of any candidate for $\text{Canonical}()$ and produce an algorithm whose run-time complexity matches it. \square

7 Inferring Typings for Small Networks

Theorem 39 (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} .⁴*

Proof Sketch. Let $\mathbf{A}_{\text{in}} = \text{in}(\mathcal{A})$ and $\mathbf{A}_{\text{out}} = \text{out}(\mathcal{A})$. To compute the interval $[r_1, r_2]$ which T assigns to a non-empty $A \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$, we carry out the following steps.

Partition A as $A = A_1 \cup A_2$ where $A_1 = A \cap \mathbf{A}_{\text{in}}$ and $A_2 = A \cap \mathbf{A}_{\text{out}}$. Let $A'_1 = \mathbf{A}_{\text{in}} - A_1$ and $A'_2 = \mathbf{A}_{\text{out}} - A_2$. Next, introduce two new “source” nodes n_{in} and n'_{in} , to originate all the input arcs in A_1 from n_{in} and all the input arcs in A'_1 from n'_{in} , *i.e.*, $n_{\text{in}} = \text{tail}(a)$ for every $a \in A_1$ and $n'_{\text{in}} = \text{tail}(a)$ for every $a \in A'_1$. Introduce two input arcs only, a_{in} and a'_{in} , one entering n_{in} and one entering n'_{in} .

Similarly, introduce two new “sink” nodes n_{out} and n'_{out} , to direct all the output arcs in A_2 to n_{out} and all the output arcs in A'_2 to n'_{out} , *i.e.*, $n_{\text{out}} = \text{head}(a)$ for every $a \in A_2$ and $n'_{\text{out}} = \text{head}(a)$ for every $a \in A'_2$. Introduce two output arcs only, a_{out} and a'_{out} , one exiting n_{out} and one exiting n'_{out} .

We set $L(a_{\text{in}}) = L(a'_{\text{in}}) = L(a_{\text{out}}) = L(a'_{\text{out}}) = 0$ and $U(a_{\text{in}}) = U(a'_{\text{in}}) = U(a_{\text{out}}) = U(a'_{\text{out}}) = \text{“a very large value”}$, *i.e.*, the new arcs a_{in} , a'_{in} , a_{out} , and a'_{out} , impose no lower bound and no upper bound on flows entering and exiting the network. Call the resulting network \mathcal{A}' .

The lower-end r_1 of the desired interval $[r_1, r_2]$ is obtained by computing: “the value of the minimum flow that must enter a_{in} ” minus “the value of the maximum flow that can exit a_{out} ”.

Similarly, the upper-end r_2 is obtained by computing: “the value of the maximum flow that can enter a_{in} ” minus “the value of the minimum flow that must exist a_{out} ”.

These values can be computed using graph theoretic ideas based on the max-cut/min-flow theorem (for the lower-end r_1) and the min-cut/max-flow theorem (for the upper-end r_2). \square

Example 40. Consider again the two small networks \mathcal{A} and \mathcal{B} from Example 13. We assign capacities to their arcs and compute their respective principal typings. The sets of arcs in \mathcal{A} and \mathcal{B} are, respectively:

$$\mathbf{A} = \{a_1, \dots, a_{11}\} \quad \text{and} \quad \mathbf{B} = \{b_1, \dots, b_{16}\}.$$

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 5 by the numbers in rectangular boxes, namely:

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

It is helpful to note that a minimal cut in \mathcal{A} consists of the set $\{a_5, a_8, a_{11}\}$, and in \mathcal{B} it consists of the set $\{b_5, b_{10}, b_{15}, b_{16}\}$, and the maximum flow across both minimal cuts is 30.

We compute the principal typings $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$. This requires the assignment of a bounded interval to every subset in $\mathcal{P}(\{a_1, a_2, a_3, a_4\})$ and $\mathcal{P}(\{b_1, b_2, b_3, b_4\})$, respectively. This is a total of 15 intervals for each, ignoring the empty set to which we assign the empty interval \emptyset . We can use the construction in the proof of Theorem 39 to compute $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$.

However, the two networks are simple enough and it is intuitively useful to compute them by inspection of

⁴For now, we leave out the issue of how efficiently we can compute T .

their respective graphs:

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

$a_1 : [0, 15]$	$a_2 : [0, 25]$	$-a_3 : [-15, 0]$	$-a_4 : [-25, 0]$
$a_1 + a_2 : [0, 30]$	$a_1 - a_3 : [-10, 10]$	$a_1 - a_4 : [-25, 15]$	
$a_2 - a_3 : [-15, 25]$	$a_2 - a_4 : [-10, 10]$	$-a_3 - a_4 : [-30, 0]$	
$a_1 + a_2 - a_3 : [0, 25]$	$a_1 + a_2 - a_4 : [0, 15]$	$a_1 - a_3 - a_4 : [-25, 0]$	$a_2 - a_3 - a_4 : [-15, 0]$
$a_1 + a_2 - a_3 - a_4 : [0, 0]$			

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

$b_1 : [0, 15]$	$b_2 : [0, 25]$	$-b_3 : [-15, 0]$	$-b_4 : [-25, 0]$
$b_1 + b_2 : [0, 30]$	$b_1 - b_3 : [-10, 12]$	$b_1 - b_4 : [-25, 15]$	
$b_2 - b_3 : [-15, 25]$	$b_2 - b_4 : [-12, 10]$	$-b_3 - b_4 : [-30, 0]$	
$b_1 + b_2 - b_3 : [0, 25]$	$b_1 + b_2 - b_4 : [0, 15]$	$b_1 - b_3 - b_4 : [-25, 0]$	$b_2 - b_3 - b_4 : [-15, 0]$
$b_1 + b_2 - b_3 - b_4 : [0, 0]$			

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, $T_{\mathcal{A}} \neq T_{\mathcal{B}}$ as well as $T_{\mathcal{A}} \neq T_{\mathcal{B}}$, the difference being in the (underlined) types assigned to some of the subsets mixing input and output arcs, specifically:

- $[-10, 10]$ 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 $\{b_2, b_4\}$.

It is not difficult to check that:

$$\text{Poly}(T_{\mathcal{A}}) = \{r_1, r_2, r_3, r_4 \in \mathbb{R}^4 \mid 0 \leq r_1, r_3 \leq 15; 0 \leq r_2, r_4 \leq 25; \\ -10 \leq r_1 - r_3 \leq 10; -10 \leq r_2 - r_4 \leq 10; 0 \leq r_1 + r_2 = r_3 + r_4 \leq 30\}$$

$$\text{Poly}(T_{\mathcal{B}}) = \{r_1, r_2, r_3, r_4 \in \mathbb{R}^4 \mid 0 \leq r_1, r_3 \leq 15; 0 \leq r_2, r_4 \leq 25; \\ -10 \leq r_1 - r_3 \leq 12; -12 \leq r_2 - r_4 \leq 10; 0 \leq r_1 + r_2 = r_3 + r_4 \leq 30\}$$

In this example, $T_{\mathcal{B}} < T_{\mathcal{A}}$ because $\text{Poly}(T_{\mathcal{A}}) \subseteq \text{Poly}(T_{\mathcal{B}})$. The converse does not hold.

Among other things, this example shows that satisfaction of the inequalities in (5) and (6) in Definition 21 does not necessarily imply satisfaction of the inequalities in (4). As a result, there are feasible flows in \mathcal{B} which are not feasible flows in \mathcal{A} . For example, if we set:

$$\begin{aligned} f_0(a_1) &= f_0(b_1) = 15 \\ f_0(a_2) &= f_0(b_2) = 0 \\ f_0(a_3) &= f_0(b_3) = 3 \\ f_0(a_4) &= f_0(b_4) = 12 \end{aligned}$$

it is easy to define a feasible flow f extending f_0 in \mathcal{B} but not in \mathcal{A} . □

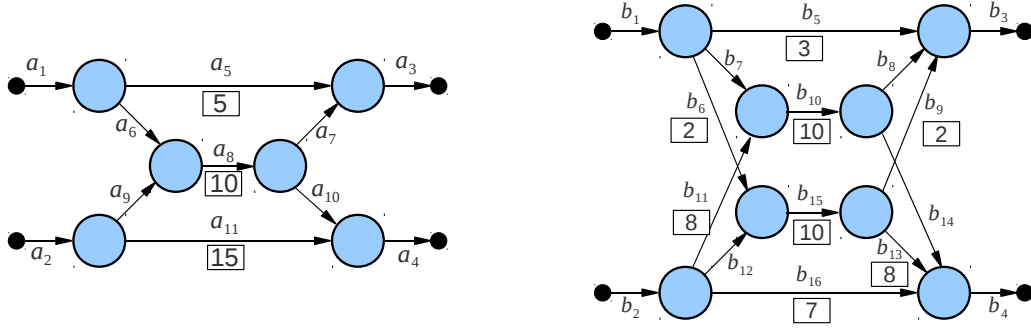


Figure 5: An assignment of arc capacities for small networks \mathcal{A} (on the left) and \mathcal{B} (on the right) in Example 40.

Example 41. In Example 40 we determined a principal typing $T_{\mathcal{A}}$ for \mathcal{A} , which is also uniformly tight and therefore unique. It is also valid and there are many other valid typings for \mathcal{A} . The following T is valid for \mathcal{A} , but not principal, and also for \mathcal{B} after the appropriate renaming of input and output arcs.

T assignments :

$$\begin{array}{llll}
 a_1 : [0, 15] & a_2 : [0, 25] & -a_3 : [-15, 0] & -a_4 : [-25, 0] \\
 a_1 + a_2 : [0, 30] & \underline{a_1 - a_3 : [-5, 5]} & a_1 - a_4 : [-25, 15] & \\
 a_2 - a_3 : [-15, 25] & \underline{a_2 - a_4 : [-5, 5]} & -a_3 - a_4 : [-30, 0] & \\
 a_1 + a_2 - a_3 : [0, 25] & a_1 + a_2 - a_4 : [0, 15] & a_1 - a_3 - a_4 : [-25, 0] & a_2 - a_3 - a_4 : [-15, 0] \\
 a_1 + a_2 - a_3 - a_4 : [0, 0] & & &
 \end{array}$$

The underlined type assignments of T are the only differences with $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$. To show that T is valid, we need to show that every flow f_0 satisfying T can be extended to a feasible flow f . To see this, consider maximal “input-skewed” flows: there are two such flows here, f_1 and f_2 , where f_1 and f_2 maximize flow through a_1 and a_2 , respectively. If f_1 is input-skewed in favor of a_1 , then $f_1(a_1) = 15$, thus forcing $f_1(a_2) = 15$, and we can easily extend f_1 to a feasible flow f'_1 in \mathcal{A} . Likewise, if f_2 is input-skewed in favor of a_2 , then $f_2(a_2) = 25$, forcing $f_2(a_1) = 5$, and we extend f_2 to a feasible flow f'_2 in \mathcal{A} . Every other flow satisfying T falls between the two extreme cases just described, corresponding to f_1 and f_2 .

Hence, T is valid for \mathcal{A} . But T cannot be principal, because there are feasible flows f in \mathcal{A} such that $f(a_1) - f(a_3) = 10$ or $f(a_2) - f(a_4) = 10$, thus violating the type $[-5, 5]$ assigned to both $\{a_1, a_3\}$ and $\{a_2, a_4\}$ by T .

Many valid typings for \mathcal{A} , and for \mathcal{B} after appropriate renaming of input and output arcs, are equivalent to *partial* typings with a far more economical assignment of as few as three type/intervals. An example of such a valid partial typing T' is the following:

T' assignments :

$$a_1 : [0, 5] \quad a_2 : [0, 10] \quad a_1 + a_2 - a_3 - a_4 : [0, 0]$$

It is easy to check that any (total) typing extending T' is valid. □

8 A Typing System

We set up a formal system for assigning typings to network specifications that are built up from small-network typings. The process of inferring typings, based on this system, is deferred to Section 9. We need several

preliminary definitions.

8.1 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 $(\mathcal{N}_1 \parallel \mathcal{N}_2)$. We define the typing $(T_1 \parallel T_2)$ for the specification $(\mathcal{N}_1 \parallel \mathcal{N}_2)$ as follows:

$$(T_1 \parallel T_2)(A) = \begin{cases} T_1(A) & \text{if } A \subseteq \mathbf{in}(\mathcal{N}_1) \cup \mathbf{out}(\mathcal{N}_1), \\ T_2(A) & \text{if } A \subseteq \mathbf{in}(\mathcal{N}_2) \cup \mathbf{out}(\mathcal{N}_2), \\ T_1(A_1) + T_2(A_2) & \text{if } A = A_1 \cup A_2 \text{ where} \\ & A_1 \subseteq \mathbf{in}(\mathcal{N}_1) \cup \mathbf{out}(\mathcal{N}_1) \text{ and } A_2 \subseteq \mathbf{in}(\mathcal{N}_2) \cup \mathbf{out}(\mathcal{N}_2). \end{cases}$$

where the operation “+” on intervals is defined in Section 6.

Lemma 42. *If $(\mathcal{N}_1 : T_1)$ and $(\mathcal{N}_2 : T_2)$ are principal typings, respectively valid typings, then so is the typing $((\mathcal{N}_1 \parallel \mathcal{N}_2) : (T_1 \parallel T_2))$ principal, respectively valid.*

Proof Sketch. Straightforward from the definitions. Details omitted. \square

Let $(\mathcal{N} : T)$ be a typing with $\langle a, b \rangle \in \mathbf{out}(\mathcal{N}) \times \mathbf{in}(\mathcal{N})$. Starting from the typing T , we explain how to define the typing we denote $\mathbf{bind}(T, \langle a, b \rangle)$ for the network specification $\mathbf{bind}(\mathcal{N}, \langle a, b \rangle)$. Suppose $\mathbf{in}(\mathcal{N}) \neq \emptyset$, $\mathbf{out}(\mathcal{N}) \neq \emptyset$, and $m = |\mathbf{in}(\mathcal{N})| + |\mathbf{out}(\mathcal{N})|$. We thus have the ordered sets:

$$\begin{aligned} \dim_{\mathbf{in}}(\mathcal{N}) &= \langle a_1, \dots, a_\ell \rangle, \\ \dim_{\mathbf{out}}(\mathcal{N}) &= \langle a_{\ell+1}, \dots, a_m \rangle, \\ \dim(\mathcal{N}) &= \dim_{\mathbf{in}}(\mathcal{N}) \cdot \dim_{\mathbf{out}}(\mathcal{N}). \end{aligned}$$

If $b = a_i$ and $a = a_j$, where $1 \leq i \leq \ell$ and $\ell + 1 \leq j \leq m$, then an equation of the form $a = b$ defines a hyperplane in the space \mathbb{R}^m , a special case of a polyhedron, which we also denote $\text{Poly}(a = b)$:

$$\text{Poly}(a = b) = \{ \langle r_1, \dots, r_m \rangle \in \mathbb{R}^m \mid r_i = r_j \}$$

(We have abused notation slightly, because we have used $\text{Poly}(\)$ to denote “polytopes” or bounded higher-dimensional polyhedra. The hyperplane defined by $a = b$ is not bounded.) Let

$$\mathbf{A}_{\mathbf{in}} = \mathbf{in}(\mathcal{N}) - \{b\} \quad \text{and} \quad \mathbf{A}_{\mathbf{out}} = \mathbf{out}(\mathcal{N}) - \{a\}$$

which are the sets of input arcs and output arcs in $\mathbf{bind}(\mathcal{N}, \langle a, b \rangle)$. We define the function

$$\mathbf{bind}(T, \langle a, b \rangle) : \mathcal{P}(\mathbf{A}_{\mathbf{in}} \cup \mathbf{A}_{\mathbf{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$$

which will be a typing for the specification $\mathbf{bind}(\mathcal{N}, \langle a, b \rangle)$ that depends on the typing T for \mathcal{N} . We need therefore to define $\mathbf{bind}(T, \langle a, b \rangle)(A)$ as an interval for every $A \subseteq \mathbf{A}_{\mathbf{in}} \cup \mathbf{A}_{\mathbf{out}}$. Consider an arbitrary such A and define a bounded set $S_A \subseteq \mathbb{R}$ as follows – we use notation from Section 5:

$$S_A = \left\{ \sum \text{Proj}_{(A \cap \mathbf{A}_{\mathbf{in}})}(\mathbf{r}) - \sum \text{Proj}_{(A \cap \mathbf{A}_{\mathbf{out}})}(\mathbf{r}) \mid \mathbf{r} \in (\mathbb{R}^+)^m \text{ and } \mathbf{r} \in \text{Poly}(T) \cap \text{Poly}(a = b) \right\}$$

If $\mathbf{s} = \langle s_1, \dots, s_n \rangle$, we write $\sum \mathbf{s}$ to denote the quantity $s_1 + \dots + s_n$. We now define:

$$\text{bind}(T, \langle a, b \rangle)(A) = \begin{cases} \emptyset & \text{if } S_A = \emptyset, \\ [\min S_A, \max S_A] & \text{otherwise,} \end{cases}$$

There is plenty of notation in the preceding for precision. More succinctly, but less explicitly, we can write:

$$\text{Poly}(\text{bind}(T, \langle a, b \rangle)) = \text{Poly}(T) \cap \text{Poly}(a = b)$$

though this does not yet assign a type (*i.e.*, an interval of reals) to every set in $\mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$. Example 44 illustrates the preceding notions on a simple small network.

Lemma 43. *If $(\mathcal{N} : T)$ is a principal (respectively, valid) typing and $\langle a, b \rangle \in \mathbf{in}(\mathcal{N}) \times \mathbf{out}(\mathcal{N})$, then $(\mathbf{bind}(\mathcal{N}, \langle a, b \rangle) : \text{bind}(T, \langle a, b \rangle))$ is a principal (respectively, valid) typing.*

Proof Sketch. Straightforward, though tedious, from the preceding construction. We omit the details. \square

Example 44. This continues our examination of small network \mathbf{M} in Examples 13, 23, and 30. \mathbf{M} is shown again in Figure 6, but now with both lower-bound and upper-bound capacities inserted:

$$\begin{array}{ll} L(d_1) = 0 & U(d_1) = 15 \\ L(d_2) = x & U(d_2) = 35 \\ L(d_3) = y & U(d_3) = 35 \end{array}$$

where x and y are set to different values to illustrate the effect of connecting output arc d_3 to input arc d_1 , which is the result of constructing $\mathbf{bind}(\mathbf{M}, \langle d_3, d_1 \rangle)$. If $x = y = 0$, the principal typing for \mathbf{M} is \tilde{T} , already mentioned in Example 23:

$$\begin{array}{lll} (i) & d_1 : [0, 15] & (ii) & d_2 : [0, 35] & (iii) & -d_3 : [-35, 0] \\ (iv) & d_1 + d_2 : [0, 35] & (v) & d_1 - d_3 : [-35, 0] & (vi) & d_2 - d_3 : [-15, 0] \\ (vii) & d_1 + d_2 - d_3 : [0, 0] & & & & \end{array}$$

If $x = 0$ and $y = 10$, a principal typing for \mathbf{M} is specified by the following interval assignment – call it \tilde{T}' :

$$\begin{array}{lll} (i) & d_1 : [0, 15] & (ii) & d_2 : [0, 35] & (iii) & -d_3 : [-35, -10] \\ (iv) & d_1 + d_2 : [10, 35] & (v) & d_1 - d_3 : [-35, 0] & (vi) & d_2 - d_3 : [-15, 0] \\ (vii) & d_1 + d_2 - d_3 : [0, 0] & & & & \end{array}$$

If $x = 5$ and $y = 0$, a principal typing for \mathbf{M} is specified by the following interval assignment – call it \tilde{T}'' :

$$\begin{array}{lll} (i) & d_1 : [0, 15] & (ii) & d_2 : [5, 35] & (iii) & -d_3 : [-35, -5] \\ (iv) & d_1 + d_2 : [5, 35] & (v) & d_1 - d_3 : [-35, 10] & (vi) & d_2 - d_3 : [-30, 0] \\ (vii) & d_1 + d_2 - d_3 : [0, 0] & & & & \end{array}$$

There is considerable redundancy in \tilde{T} , \tilde{T}' , and \tilde{T}'' , in that many of the type assignments can be removed without changing the meaning of $\text{Poly}(\tilde{T})$, $\text{Poly}(\tilde{T}')$, and $\text{Poly}(\tilde{T}'')$. We will not worry about this redundancy here – we already examined how to remove it in the case of \tilde{T} in Example 23 and left the general case as Open Problem 38. Graphical representations are shown in Figure 7: The three lightly shaded surfaces are precisely $\text{Poly}(\tilde{T})$, $\text{Poly}(\tilde{T}')$, and $\text{Poly}(\tilde{T}'')$, from left to right, respectively.

By Lemma 43, if T is a principal typing for \mathbf{M} and $\text{Poly}(T) \cap \text{Poly}(d_3 = d_1) \neq \emptyset$, then $\text{bind}(T, \langle d_3, d_1 \rangle)$ is a principal typing for $\mathbf{bind}(\mathbf{M}, \langle d_3, d_1 \rangle)$. It is readily checked that:

$$\text{Poly}(\tilde{T}) \cap \text{Poly}(d_3 = d_1) \neq \emptyset$$

$$\text{Poly}(\tilde{T}') \cap \text{Poly}(d_3 = d_1) \neq \emptyset$$

$$\text{Poly}(\tilde{T}'') \cap \text{Poly}(d_3 = d_1) = \emptyset$$

Hence, when $x = y = 0$ (resp. when $x = 0$ and $y = 10$), $\text{bind}(\tilde{T}, \langle d_3, d_1 \rangle)$ is a principal typing (resp. $\text{bind}(\tilde{T}', \langle d_3, d_1 \rangle)$ is a principal typing) for $\mathbf{bind}(\mathbf{M}, \langle d_3, d_1 \rangle)$. On the other hand, when $x = 5$ and $y = 0$, there is no feasible flow in $\mathbf{bind}(\mathbf{M}, \langle d_3, d_1 \rangle)$ and its principal typing is $\text{bind}(\tilde{T}'', \langle d_3, d_1 \rangle) = \emptyset$. \square

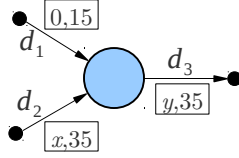


Figure 6: An assignment of lower-bound and upper-bound capacities for the small network \mathbf{M} in Example 44.

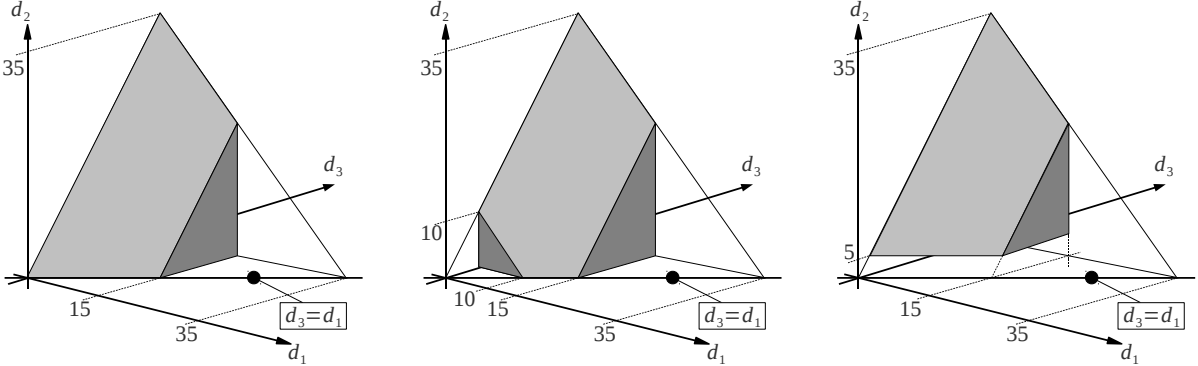


Figure 7: From Example 44, $\text{Poly}(\tilde{T})$, $\text{Poly}(\tilde{T}')$, and $\text{Poly}(\tilde{T}'')$, are shown as light-shaded surfaces, on the left, in the middle, and on the right, respectively. The two first intersect $\text{Poly}(d_3 = d_1)$, the third does not.

8.2 Typing Rules

The system is in Figure 8, where we follow standard conventions in formulating the rules. We call Γ a *typing environment* (or *context*), which is a finite set of *typing assumptions* for holes, each of the form $(X : T)$. If $(X : T)$ is a typing assumption, with $\text{in}(X) = \mathbf{A}_{\text{in}}$ and $\text{out}(X) = \mathbf{A}_{\text{out}}$, then $T : \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$.

In the rule LET, assumptions are discharged from the context Γ . This is not essential, because we assume there is at most one binding occurrence for every hole in a network specification and we purposely avoid any process of “reducing” a network specification whereby all let-bindings have been eliminated. (Review the conditions for well-formedness in Section 3.2 to back up these comments). We discharge assumptions in the rule LET for conciseness and only to indicate which holes in a network specification remain unbound.

If a typing T is derived for a network specification \mathcal{N} according to the rules in Figure 8, 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. The side conditions in Figure 8 must be satisfied in order that the corresponding rules can be applied.

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

Figure 8: Typing Rules for Flow Networks. The operations $(T_1 \parallel T_2)$ and $\mathbf{bind}(T, \langle a, b \rangle)$ are defined in Section 8.1.

Theorem 45 (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 8, 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} .

Proof Sketch. Straightforward induction, using Lemmas 42 and 43. □

9 Inferring Typings for Flow Networks in General

First consider the case when \mathcal{N} contains no let-binding. This means that, starting from a finite collection of small networks $\{\mathcal{A}_1, \dots, \mathcal{A}_m\}$, the specification \mathcal{N} is assembled by applying several times the “ \parallel ” and “**bind**” constructors in some order. Suppose the intermediate specifications we need to define before \mathcal{N} is fully produced are: $\mathcal{M}_1, \dots, \mathcal{M}_n$. Each small network \mathcal{A}_i appears as some \mathcal{M}_j in this collection, and \mathcal{N} is the last \mathcal{M}_n in this collection. For such a let-free specification \mathcal{N} we can proceed according to one of two methods, starting in both cases from the principal typings $\{T_{\mathcal{A}_1}, \dots, T_{\mathcal{A}_n}\}$ of the small networks:

method 1 We collect symbolically all intermediate typings, resulting in a symbolic expression denoting T built up using several times the (typing) constructors “ \parallel ” and “**bind**” in some order, following the inductive definition of \mathcal{N} . We solve for T , i.e., determine the interval $T(A)$ for every $A \subseteq \mathbf{in}(\mathcal{N}) \cup \mathbf{out}(\mathcal{N})$, and also decide whether $\text{Poly}(T) \neq \emptyset$, after the construction of \mathcal{N} is completed.

method 2 We compute the interval $T_i(A)$ for every $A \subseteq \mathbf{in}(\mathcal{M}_i) \cup \mathbf{out}(\mathcal{M}_i)$ and decide whether $\text{Poly}(T_i) \neq \emptyset$, where T_i is a typing for \mathcal{M}_i for $i = 1, \dots, n$, *simultaneously* with the construction of \mathcal{N} at every intermediate step in the latter’s inductive definition. We stop the procedure from the moment $\text{Poly}(T_i) = \emptyset$ for some $1 \leq i \leq n$, or else determine a non-empty typing for $\mathcal{M}_n = \mathcal{N}$.

In a sense, both methods do the same thing. But the organization and book-keeping parts of the two algorithms are different. In both methods, a typing induced by the constructor “ \parallel ” is straightforward to generate, but not that by the constructor “**bind**”, and each method handles the latter differently.

In **method 1**, from the symbolic expression denoting the final typing T for \mathcal{N} , we generate a finite set of linear constraints and, for each $A \subseteq \mathbf{in}(\mathcal{N}) \cup \mathbf{out}(\mathcal{N})$, optimize (minimize and maximize) the quantity:

$$S = \sum(A \cap \mathbf{in}(\mathcal{N})) - \sum(A \cap \mathbf{out}(\mathcal{N}))$$

relative to these constraints using linear programming. We assign to $T(A)$ the interval $[\min S, \max S]$. This is illustrated in Example 48.

In **method 2**, we invoke one of two algorithms PARTYPING or BINDTYPING at every step, depending on whether the last-used constructor is “ \parallel ” or “**bind**”. This has the advantage of allowing to stop the procedure from the moment $\text{Poly}(T_i) = \emptyset$, where T_i is the typing of an intermediate network specification in the induction, but with the drawback of incurring an extra cost at every step.

Algorithms PARTYPING and BINDTYPING are shown in Figures 9 and 10. Example 48 illustrates **method 2** and its use of algorithms PARTYPING and BINDTYPING.

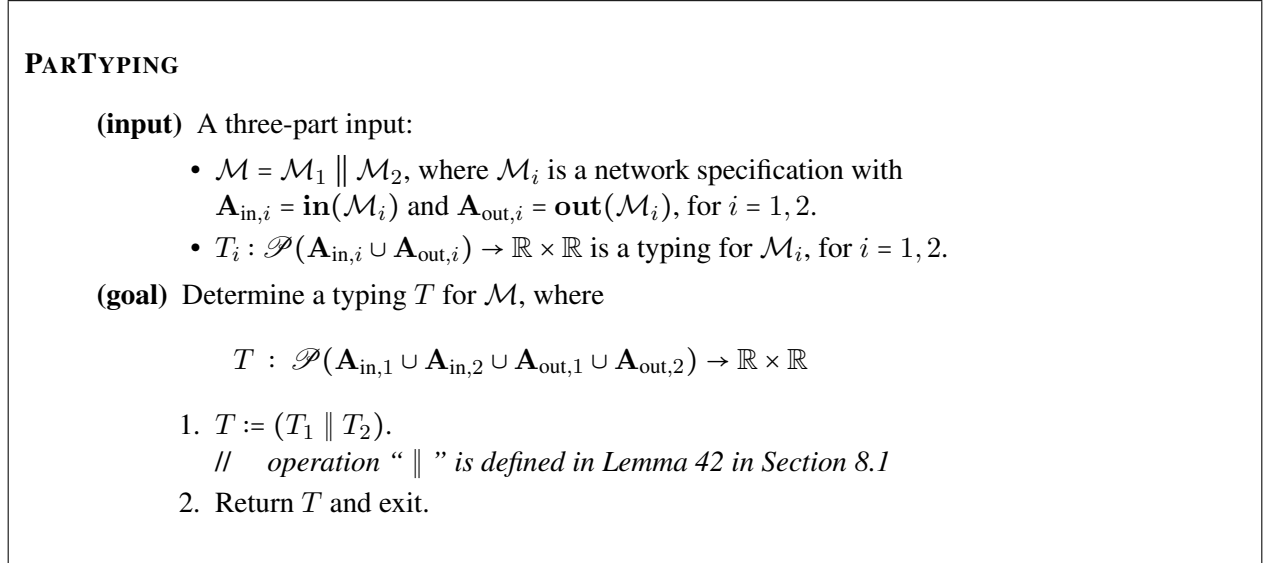


Figure 9: Algorithm PARTYPING.

Proposition 46 (Correctness of Algorithm PARTYPING). *Let $\mathcal{M} = (\mathcal{M}_1 \parallel \mathcal{M}_2)$, and let T_1 and T_2 be valid (resp. principal) typings for \mathcal{M}_1 and \mathcal{M}_2 , respectively. Then PARTYPING always terminates and returns a valid (resp. principal) typing for \mathcal{M} .*

Proof Sketch. This follows from Lemma 42. □

Proposition 47 (Correctness of Algorithm BINDTYPING). *Let $\mathcal{M} = \mathbf{bind}(\mathcal{P}, \langle a, b \rangle)$ and T a valid (resp. principal) typing for \mathcal{P} . Then BINDTYPING always stops and returns a valid (resp. principal) typing for \mathcal{M} .*

Proof Sketch. This follows from Lemma 43. □

Example 48. Consider the two small networks \mathcal{A} and \mathcal{B} , and their principal typings $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$, from Example 40. We want to infer a principal typing for the network specification:

$$\mathcal{N} = \mathbf{bind} \left(\left(\mathbf{bind}(\mathcal{A}, \langle a_3, a_2 \rangle) \parallel \mathbf{bind}(\mathcal{B}, \langle b_3, b_2 \rangle) \right), \langle a_4, b_1 \rangle \right)$$

According to **method 1**, we complete the construction of \mathcal{N} first, based on which we can symbolically write the resulting typing of \mathcal{N} , which is here:

$$T_{\mathcal{N}} = \mathbf{bind}(\langle a_4, b_1 \rangle, (\mathbf{bind}(\langle a_3, a_2 \rangle, T_{\mathcal{A}}) \parallel \mathbf{bind}(\langle b_3, b_2 \rangle, T_{\mathcal{B}})))$$

BINDTYPING

(input) A two-part input:

- $\mathcal{M} = \mathbf{bind}(\mathcal{P}, \langle a, b \rangle)$ where $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{P})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{P})$ and $\langle a, b \rangle \in \mathbf{A}_{\text{out}} \times \mathbf{A}_{\text{in}}$.
- $T : \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$ is a typing for \mathcal{P} .

(goal) Determine a typing T' for \mathcal{M} , where

$$T' : \mathcal{P}((\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) - \{a, b\}) \rightarrow \mathbb{R} \times \mathbb{R}$$

1. $T' := \mathbf{bind}(T, \langle a, b \rangle)$.
// operation “bind” is defined in Lemma 43 in Section 8.1
2. $T' := \mathbf{Tight}(T')$.
// T' obtained in step 1 is not necessarily uniformly tight,
// e.g., if we did not apply $\mathbf{Tight}()$ to the typing of \mathcal{M}_1 in Example 48,
// the interval assigned to $\{a_4\}$ would be $[-25, 0]$ instead of $[-15, 0]$
3. Return T' and exit.

Figure 10: Algorithm BINDTYPING.

It is easy to see that $\mathbf{in}(\mathcal{N}) = \{a_1\}$ and $\mathbf{out}(\mathcal{N}) = \{b_4\}$. Hence, to determine $T_{\mathcal{N}}$ we need to assign intervals/types to each of $T_{\mathcal{N}}(\{a_1\})$, $T_{\mathcal{N}}(\{b_4\})$, and $T_{\mathcal{N}}(\{a_1, b_4\})$. We do not need to bother about $T_{\mathcal{N}}(\emptyset)$, to which we always assign the empty interval, nor about $T_{\mathcal{N}}(\{a_1, b_4\})$ to which we assign the interval $[0, 0]$.

The only non-trivial types are $T_{\mathcal{N}}(\{a_1\})$ and $T_{\mathcal{N}}(\{b_4\})$, and in fact $T_{\mathcal{N}}(\{a_1\}) = -T_{\mathcal{N}}(\{b_4\})$, so we only need to determine one (the first, say). For this, we collect the inequalities induced by $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$, according to (9) in Section 5.1, and add the 3 equality constraints induced by the 3 uses of the constructor **bind**:

$$\begin{aligned} & \{T_{\mathcal{A}, \geq}(A) \mid \emptyset \neq A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})\} \cup \{T_{\mathcal{A}, \leq}(A) \mid \emptyset \neq A \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})\} \cup \\ & \{T_{\mathcal{B}, \geq}(B) \mid \emptyset \neq B \in \mathcal{P}(\mathbf{B}_{\text{in}} \cup \mathbf{B}_{\text{out}})\} \cup \{T_{\mathcal{B}, \leq}(B) \mid \emptyset \neq B \in \mathcal{P}(\mathbf{B}_{\text{in}} \cup \mathbf{B}_{\text{out}})\} \cup \\ & \{a_3 = a_2\} \cup \{b_3 = b_2\} \cup \{a_4 = b_1\} \end{aligned}$$

We next determine the smallest possible value r and the largest possible value r' which we can assign to a_1 without violating these constraints. The desired interval/type will be $T_{\mathcal{N}}(\{a_1\}) = [r, r']$. We omit the remaining details of **method 1**, which can be supplied using any of the well-known algorithms for linear programming.

In **method 2** we gradually determine the relevant intervals/types at every step of the inductive definition of \mathcal{N} . Let T_1, T_2 , and T_3 be the yet-to-be-determined typings of:

$$\begin{aligned} \mathcal{M}_1 &= \mathbf{bind}(\mathcal{A}, \langle a_3, a_2 \rangle), \\ \mathcal{M}_2 &= \mathbf{bind}(\mathcal{B}, \langle b_3, b_2 \rangle), \\ \mathcal{M}_3 &= (\mathbf{bind}(\mathcal{A}, \langle a_3, a_2 \rangle) \parallel \mathbf{bind}(\mathcal{B}, \langle b_3, b_2 \rangle)), \end{aligned}$$

respectively. We determine T_1 and T_2 from $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$, respectively, using algorithm BINDTYPING. It is a straightforward computation:

$$\begin{aligned} T_1(\{a_1\}) &= -T_1(\{a_4\}) = [0, 15], & \text{and} & \quad T_1(\{a_1, a_4\}) = [0, 0], \\ T_2(\{b_1\}) &= -T_2(\{b_4\}) = [0, 15], & \text{and} & \quad T_2(\{b_1, b_4\}) = [0, 0]. \end{aligned}$$

We determine T_3 from T_1 and T_2 using algorithm PARTYPING. Again, it is a straightforward computation. T_3 includes all the interval assignments of T_1 and T_2 , in addition to interval assignments to sets mixing input/output arcs from \mathcal{M}_1 and \mathcal{M}_2 :

$$T_3(\{a_1, b_1\}) = -T_3(\{a_4, b_4\}) = [0, 30], \quad T_3(\{a_1, b_4\}) = T_3(\{b_1, a_4\}) = [-15, 15], \dots$$

where we omit the other straightforward interval assignments. Finally, $T_{\mathcal{N}}$ is obtained from T_3 using algorithm BINDTYPING:

$$T_{\mathcal{N}}(\{a_1\}) = -T_{\mathcal{N}}(\{b_4\}) = [0, 15], \quad \text{and} \quad T_{\mathcal{N}}(\{a_1, b_4\}) = [0, 0].$$

Having started from principal typings $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$, the resulting typing $T_{\mathcal{N}}$ is principal for \mathcal{N} . \square

Typing inference in the presence of let-bindings

Consider a specification \mathcal{N} of the form **(let $X = \mathcal{M}$ in \mathcal{P})**. Let $\mathbf{A}_{\text{in}} = \text{in}(X)$ and $\mathbf{A}_{\text{out}} = \text{out}(X)$. Suppose X occurs $n \geq 1$ times in \mathcal{P} , so that its input/output arcs are renamed in each of the n occurrences according to:

$${}^1(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}), \dots, {}^n(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}).$$

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:

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

for every $1 \leq i \leq n$. According to the typing rules in Figure 8, a valid typing for \mathcal{N} requires that:

$$T_X \approx {}^1 T_X \approx \dots \approx {}^n T_X.$$

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 ${}^i T_X(B)$ and every $B \subseteq {}^i \mathbf{A}_{\text{in}} \cup {}^i \mathbf{A}_{\text{out}}$. For later reference, call x_B a *lower-end parameter* and y_B an *upper-end parameter*. 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 ${}^1 T_{\mathcal{M}}, \dots, {}^n T_{\mathcal{M}}$ 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 a principal typing $T_{\mathcal{M}}$ for \mathcal{M} , and a principal typing $T_{\mathcal{P}}$ for \mathcal{P} , separately. $T_{\mathcal{P}}$ is now parametrized by the typings ${}^i T_X$ written symbolically. A final typing for \mathcal{N} is obtained by setting the lower-end and upper-end parameters in ${}^i T_X$ to the 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.*, \mathcal{M} first, or \mathcal{P} first, or both simultaneously). 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 understand. We delay the examination of the parallel/fully-compositional approach to a follow-up report.

An implementation of the sequential approach is algorithm SEQINFERENCE, in Figure 11, where we use the three following notational conventions – in steps 3, 4, and 5, respectively:

- (a) $\text{PARTYPING}(\mathcal{N}, T_1, T_2)$ denotes an activation of algorithm PARTYPING , provided \mathcal{N} is of the form $(\mathcal{M}_1 \parallel \mathcal{M}_2)$ and T_i is a typing for \mathcal{M}_i for $i = 1, 2$.
- (b) $\text{BINDTYPING}(\mathcal{N}, T)$ denotes an activation of algorithm BINDTYPING , provided \mathcal{N} is of the form $\mathbf{bind}(\mathcal{P}, \langle a, b \rangle)$ and T is a typing for \mathcal{P} .
- (c) Suppose X is a hole with $\mathbf{A}_{\text{in}} = \mathbf{in}(X) = \{a_1, \dots, a_k\}$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(X) = \{a_{k+1}, \dots, a_{k+\ell}\}$. Suppose a typing T_X for X is yet to be defined, and that T is a typing over the same dimension as X , *i.e.*,

$$T : \mathcal{P}(\mathbf{B}_{\text{in}} \cup \mathbf{B}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R}$$

where $\mathbf{B}_{\text{in}} = \{b_1, \dots, b_k\}$ and $\mathbf{B}_{\text{out}} = \{b_{k+1}, \dots, b_{k+\ell}\}$. We write $T_X \approx T$ to indicate that T_X is assigned the typing T after appropriate renaming, *i.e.*,

$$\begin{aligned} T_X &: \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathbb{R} \times \mathbb{R} \\ T_X(\{a_{i_1}, \dots, a_{i_m}\}) &:= T(\{b_{i_1}, \dots, b_{i_m}\}) \end{aligned}$$

for every $\{i_1, \dots, i_m\} \subseteq \{1, \dots, k + \ell\}$.

Algorithm SEQINFERENCE is recursive and uses a global set \mathcal{T} of typings. An activation of SEQINFERENCE starts when it receives a network specification \mathcal{N} as input, which is thus denoted $\text{SEQINFERENCE}(\mathcal{N})$. The global set \mathcal{T} contains a typing $T_{\mathcal{A}}$ for every small network \mathcal{A} occurring in \mathcal{N} as well as a typing T_X for every hole X with *free* occurrences in \mathcal{N} .

Proposition 49 (Correctness of Algorithm SEQINFERENCE). *Let \mathcal{N} be a closed network specification, *i.e.*, there is a let-binding for every hole X occurring in \mathcal{N} . For every small network \mathcal{A} occurring in \mathcal{N} , let \mathcal{T} contain a valid (resp. principal) typing $T_{\mathcal{A}}$ for \mathcal{A} . Then $\text{SEQINFERENCE}(\mathcal{N})$ always terminates and returns a valid (resp. principal) typing for \mathcal{N} .*

Proof Sketch. This follows from Proposition 46 and 47. □

Remark 50. If \mathcal{N} contains no let-binding, $\text{SEQINFERENCE}(\mathcal{N})$ will consist of several uses of PARTYPING and BINDTYPING only, *i.e.*, only steps 1, 3, and 4, in SEQINFERENCE will be used. In this case, the execution of SEQINFERENCE thus proceeds according to what we call **method 2** in the opening paragraph of Section 9.

It is possible to define a version of SEQINFERENCE so that its execution corresponds to **method 1**, whereby constraints are accumulated symbolically until all of \mathcal{N} has been processed. In this version of SEQINFERENCE , resolving the constraints is delayed to the end of its execution. □

10 Semantics of Flow Networks Relative to Objective Functions

Let \mathcal{N} be a network, with $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$, $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$, $\mathbf{A}_{\#} = \#\mathcal{N}$, and:

$$\mathbf{A} = \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \cup \mathbf{A}_{\#}$$

We write $\mathbf{A}_{\text{out},\#}$ to denote $\mathbf{A}_{\text{out}} \cup \mathbf{A}_{\#}$, the set of all arcs in \mathcal{N} excluding the input arcs. An *objective function* selects a subset of feasible flows $f \in \llbracket \mathcal{N} \rrbracket$ that satisfy the minimization (or maximization) of some quantity. We list three possible objective functions, among several others, commonly considered in “traffic engineering” (see [BL06] for example).

Minimize Hop Routing (HR) A minimum hop route is a route with minimal number of links.

Given a feasible flow $f \in \llbracket \mathcal{N} \rrbracket$, we define the quantity $\text{HR}(f) = \sum_{a \in \mathbf{A}_{\text{out},\#}} f(a)$. Given two feasible flows $f_1, f_2 \in \llbracket \mathcal{N} \rrbracket$, we write $f_1 <^{\text{HR}} f_2$ iff two conditions:

SEQINFERENCE

(input) Network specification \mathcal{N} .

(global variable) \mathcal{T} is a set of typings, containing a typing $T_{\mathcal{A}}$ for every small network \mathcal{A} occurring in \mathcal{N} and a typing T_X for every hole X occurring *free* in \mathcal{N} .

(goal) Determine a typing T for \mathcal{N} .

Compute T recursively, *i.e.*, by induction on the structure of \mathcal{N} :

1. If \mathcal{N} is the small network \mathcal{A} , then $T := T_{\mathcal{A}} \in \mathcal{T}$.
2. If \mathcal{N} is the renamed occurrence iX of hole X , then $T := {}^iT_X$, where iT_X is obtained from $T_X \in \mathcal{T}$ by appropriate renaming of its dimensions.
3. If \mathcal{N} is $(\mathcal{M}_1 \parallel \mathcal{M}_2)$, then

$$\begin{aligned} T_1 &:= \text{SEQINFERENCE}(\mathcal{M}_1), \\ T_2 &:= \text{SEQINFERENCE}(\mathcal{M}_2), \\ T &:= \text{PARTYPING}(\mathcal{N}, T_1, T_2). \end{aligned}$$

4. If \mathcal{N} is **bind** $(\mathcal{P}, \langle a, b \rangle)$, then

$$\begin{aligned} T' &:= \text{SEQINFERENCE}(\mathcal{P}), \\ T &:= \text{BINDTYPING}(\mathcal{N}, T'). \end{aligned}$$

5. If \mathcal{N} is **(let $X = \mathcal{M}$ in \mathcal{P})**, then

$$\begin{aligned} T_X &\approx \text{SEQINFERENCE}(\mathcal{M}), \\ \mathcal{T} &:= \mathcal{T} \cup \{T_X\}, \\ T &:= \text{SEQINFERENCE}(\mathcal{P}). \end{aligned}$$

6. Return T and exit.

Figure 11: Algorithm SEQINFERENCE.

- $[f_1]_{\mathbf{A}_{\text{in}}} = [f_2]_{\mathbf{A}_{\text{in}}}$, 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 $f_1 \leq^{\text{HR}} f_2$ 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 $f \in \llbracket \mathcal{N} \rrbracket$, we define the quantity $\text{AU}(f) = \sum_{a \in \mathbf{A}_{\text{out},\#}} u(a)$. Given two feasible flows $f_1, f_2 \in \llbracket \mathcal{N} \rrbracket$, 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:

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

Hence, 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 $f_1 \leq^{\text{AU}} f_2$ to mean $f_1 <^{\text{AU}} f_2$ or $\text{AU}(f_1) = \text{AU}(f_2)$.

Minimize Mean Delay (MD) The mean delay of an arc a can be measured by $d(a) = 1/(U(a) - f(a))$.

Given a feasible flow $f \in \llbracket \mathcal{N} \rrbracket$, we define the quantity $\text{MD}(f) = \sum_{a \in \mathbf{A}_{\text{out},\#}} d(a)$. Given two feasible flows $f_1, f_2 \in \llbracket \mathcal{N} \rrbracket$, we write $f_1 <^{\text{MD}} f_2$ iff two conditions:

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

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

$$\sum \{ 1/(U(a) - f_1(a))^2 \mid a \in \mathbf{A}_{\text{out},\#} \} < \sum \{ 1/(U(a) - f_2(a))^2 \mid a \in \mathbf{A}_{\text{out},\#} \}$$

In contrast to **HR** and **AU**, the minimization of **MD** depends on the flow carried by the arcs in $\mathbf{A}_{\text{out},\#}$. We write $f_1 \leq^{\text{MD}} f_2$ to mean $f_1 <^{\text{MD}} f_2$ or $\text{MD}(f_1) = \text{MD}(f_2)$.

Remark 51. The definition of the functions **HR**, **AU**, and **MD**, is a summation over $\mathbf{A}_{\text{out},\#} = \mathbf{A}_{\text{out}} \cup \mathbf{A}_{\#}$. An alternative is to sum over $\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\#}$ or over only $\mathbf{A}_{\#}$. There are minor technical differences between these three alternatives. Our choice for summing over $\mathbf{A}_{\text{out},\#}$ simplifies a little clause 5, in the formal semantics below. \square

For the rest of this section, consider a fixed objective $\alpha \in \{\text{HR}, \text{AU}, \text{MD}, \dots\}$. 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* of the form $\sum_{a \in \mathbf{A}_{\text{out},\#}} \theta(a)$ for some $\theta : \mathbf{A}_{\text{out},\#} \rightarrow \mathbb{R}^+$. The three particular objectives considered above are all additive aggregate.

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

- $f \in \llbracket \mathcal{N} \rrbracket$, i.e., f is a feasible flow in \mathcal{N} ,
- $B \subseteq \text{in}(\mathcal{N}) \cup \text{out}(\mathcal{N})$,
- $r = \alpha(f)$,

such that, for every feasible flow $g \in \llbracket \mathcal{N} \rrbracket$, if $[f]_B = [g]_B$ then $\alpha(g) \geq r$. The extra information provided by the parameters B and r allows us to push the induction through in a compositional manner, in clause 5 in the definition of $\llbracket \mathcal{N} \mid \alpha \rrbracket$ 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 $\langle f, B, r \rangle \in \llbracket \mathcal{N} \mid \alpha \rrbracket$, then among all feasible flows that agree on B , flow f minimizes $\alpha(f)$. We include the parameter $r = \alpha(f)$ in the triple in order 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:

$$\begin{aligned} \llbracket \mathcal{A} \mid \alpha \rrbracket = \{ \langle f, B, r \rangle \mid f \in \llbracket \mathcal{A} \rrbracket, B \subseteq \text{in}(\mathcal{A}) \cup \text{out}(\mathcal{A}), r = \alpha(f), \\ \text{and for every } g \in \llbracket \mathcal{A} \rrbracket, \text{ if } [f]_B = [g]_B \text{ then } \alpha(f) \leq \alpha(g) \} \end{aligned}$$

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

$$\langle\langle \mathcal{A} | \alpha \rangle\rangle = \{ \langle [f]_A, B, r \rangle \mid \langle f, B, r \rangle \in \llbracket \mathcal{A} | \alpha \rrbracket \}$$

where $A = \mathbf{in}(\mathcal{A}) \cup \mathbf{out}(\mathcal{A})$.

As in Section 4, the full semantics $\llbracket X | \alpha \rrbracket$ 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 $\mathbf{A}_{\text{out}} = \mathbf{out}(X)$, and r, r' real numbers such that $0 \leq r \leq r'$. Then:

$$\begin{aligned} \llbracket X | \alpha \rrbracket = \langle\langle X | \alpha \rangle\rangle \subseteq \{ \langle f, B, s \rangle \mid f : \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \rightarrow \mathbb{R}^+, B \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}, s \in \mathbb{R}^+, \\ \text{and } r \leq \sum f(\mathbf{A}_{\text{in}}) = \sum f(\mathbf{A}_{\text{out}}) \leq r' \} \end{aligned}$$

Again, as in Section 4, $\llbracket X | \alpha \rrbracket = \langle\langle X | \alpha \rangle\rangle$ 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 $\llbracket \mathcal{M} | \alpha \rrbracket$ for every subexpression \mathcal{M} of \mathcal{N} , and simultaneously check the conditions that $\llbracket X | \alpha \rrbracket$ is *well-defined*, by induction on the structure of the specification \mathcal{N} . At the end, we define $\langle\langle X | \alpha \rangle\rangle$ from $\llbracket X | \alpha \rrbracket$. 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 52 establishes the correctness of this definition:

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

$$\llbracket \mathcal{M} | \alpha \rrbracket = \{ \langle f_1 \parallel f_2, B_1 \cup B_2, r_1 + r_2 \rangle \mid \langle f_1, B_1, r_1 \rangle \in \llbracket \mathcal{P}_1 | \alpha \rrbracket \text{ and } \langle f_2, B_2, r_2 \rangle \in \llbracket \mathcal{P}_2 | \alpha \rrbracket \}$$

4. If $\mathcal{M} = (\mathbf{let } X = \mathcal{P} \mathbf{ in } \mathcal{P}')$, then $\llbracket \mathcal{M} | \alpha \rrbracket = \llbracket \mathcal{P}' | \alpha \rrbracket$, provided two conditions:

(a) $\dim(X) \approx \dim(\mathcal{P})$,

(b) for every $f : \mathbf{in}(X) \cup \mathbf{out}(X) \rightarrow \mathbb{R}^+$, $B \subseteq \mathbf{in}(X) \cup \mathbf{out}(X)$, and $r \in \mathbb{R}^+$,

$$\langle f, B, r \rangle \in \llbracket X | \alpha \rrbracket \quad \text{iff} \quad \text{there is } \langle g, C, r \rangle \in \llbracket \mathcal{P} | \alpha \rrbracket \text{ such that } f \approx [g]_A \text{ and } B \approx C,$$

where $A = \mathbf{in}(\mathcal{P}) \cup \mathbf{out}(\mathcal{P})$.

5. If $\mathcal{M} = \mathbf{bind}(\mathcal{P}, \langle a, b \rangle)$, then

$$\begin{aligned} \llbracket \mathcal{M} | \alpha \rrbracket = \{ \langle f, B, r \rangle \mid \langle f, B \cup \{a, b\}, r \rangle \in \llbracket \mathcal{P} | \alpha \rrbracket, f(a) = f(b), \\ \text{and for every } \langle g, B \cup \{a, b\}, s \rangle \in \llbracket \mathcal{P} | \alpha \rrbracket \\ \text{if } g(a) = g(b) \text{ and } [f]_B = [g]_B \text{ then } r \leq s \} \end{aligned}$$

Note that, even though $[f]_B = [g]_B$, in general $f(a) = f(b) \neq g(a) = g(b)$.

All of Remarks 14, 15, and 16 in Section 4, apply again here, properly relativized to the objective α . We now define $\langle\langle \mathcal{N} | \alpha \rangle\rangle$ from $\llbracket \mathcal{N} | \alpha \rrbracket$:

$$\langle\langle \mathcal{N} | \alpha \rangle\rangle = \{ \langle [f]_A, B, r \rangle \mid \langle f, B, r \rangle \in \llbracket \mathcal{N} | \alpha \rrbracket \}$$

where $A = \mathbf{in}(\mathcal{N}) \cup \mathbf{out}(\mathcal{N})$.

Proposition 52 (Correctness of Flow-Network Semantics, Relativized). *Let \mathcal{N} be a network specification and let α be an additive aggregate objective. For every $f : \mathbf{A}_{in} \cup \mathbf{A}_{out} \cup \mathbf{A}_{\#} \rightarrow \mathbb{R}^+$, every $B \subseteq \mathbf{A}_{in} \cup \mathbf{A}_{out}$, and every $r \in \mathbb{R}^+$, it is the case that:*

$$\langle f, B, r \rangle \in \llbracket \mathcal{N} \mid \alpha \rrbracket \quad \text{iff} \quad f \in \llbracket \mathcal{N} \rrbracket \text{ and } r = \alpha(f) \text{ and} \\ \text{for every } g \in \llbracket \mathcal{N} \rrbracket, \text{ if } [f]_B = [g]_B \text{ then } \alpha(g) \geq r.$$

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

Proof. The proof is by induction on the definition of \mathcal{N} . To push the induction through, we need to strengthen the induction hypothesis. The strengthened induction hypothesis (IH) will read as follows, where $\llbracket \mathcal{N} \mid \alpha \rrbracket$ is a set of quadruples to be defined yet:

(IH) For every $f : \mathbf{A}_{in} \cup \mathbf{A}_{out} \cup \mathbf{A}_{\#} \rightarrow \mathbb{R}^+$, every $B \subseteq \mathbf{A}_{in} \cup \mathbf{A}_{out}$, every $\mathcal{C} \subseteq \mathbf{A}_{in} \times \mathbf{A}_{out}$, and every $r \in \mathbb{R}^+$, it is the case that:

$$\langle f, B, \mathcal{C}, r \rangle \in \llbracket \mathcal{N} \mid \alpha \rrbracket \quad \text{iff} \quad f \in \llbracket \mathcal{N} \rrbracket, f \models \mathcal{C}, \text{ and } r = \alpha(f), \text{ and} \\ \text{for every } g \in \llbracket \mathcal{N} \rrbracket, \text{ if } [f]_B = [g]_B \text{ and } g \models \mathcal{C} \text{ then } \alpha(g) \geq r.$$

We write \mathcal{C} as a set of equalities, say $\{a_1 = a'_1, \dots, a_k = a'_k\}$ where $\{a_1, \dots, a_k\} \subseteq \mathbf{A}_{in}$ and $\{a'_1, \dots, a'_k\} \subseteq \mathbf{A}_{out}$, and write $f \models \mathcal{C}$ iff $f(a_i) = f(a'_i)$ for every $1 \leq i \leq k$. We give the full details of the inductive definition of $\llbracket \mathcal{N} \mid \alpha \rrbracket$. With every small network \mathcal{A} , we set:

1. $\llbracket \mathcal{A} \mid \alpha \rrbracket = \left\{ \langle f, B, \mathcal{C}, r \rangle \mid f \in \llbracket \mathcal{A} \rrbracket, B \subseteq \mathbf{in}(\mathcal{A}) \cup \mathbf{out}(\mathcal{A}), \mathcal{C} \subseteq \mathbf{in}(\mathcal{A}) \times \mathbf{out}(\mathcal{A}), \right. \\ \left. f \models \mathcal{C}, r = \alpha(f), \text{ and for every } g \in \llbracket \mathcal{A} \rrbracket, \right. \\ \left. \text{if } [f]_B = [g]_B \text{ and } g \models \mathcal{C}, \text{ then } \alpha(f) \leq \alpha(g) \right\}$

For a hole X , let $\mathbf{A}_{in} = \mathbf{in}(X)$ and $\mathbf{A}_{out} = \mathbf{out}(X)$, and r, r' real numbers such that $0 \leq r \leq r'$. We set:

2. $\llbracket X \mid \alpha \rrbracket \subseteq \left\{ \langle f, B, \mathcal{C}, s \rangle \mid f : \mathbf{A}_{in} \cup \mathbf{A}_{out} \rightarrow \mathbb{R}^+, B \subseteq \mathbf{A}_{in} \cup \mathbf{A}_{out}, \mathcal{C} \subseteq \mathbf{A}_{in} \times \mathbf{A}_{out} \right. \\ \left. f \models \mathcal{C}, s \in \mathbb{R}^+, \text{ and } r \leq \sum f(\mathbf{A}_{in}) = \sum f(\mathbf{A}_{out}) \leq r' \right\}$

The rest of the induction proceeds as follows:

3. If $\mathcal{M} = (\mathcal{P}_1 \parallel \mathcal{P}_2)$, then

$$\llbracket \mathcal{M} \mid \alpha \rrbracket = \left\{ \langle f_1 \parallel f_2, B_1 \cup B_2, \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}, r_1 + r_2 \rangle \mid \right. \\ \langle f_1, B_1, \mathcal{C}_1, r_1 \rangle \in \llbracket \mathcal{P}_1 \mid \alpha \rrbracket, \langle f_2, B_2, \mathcal{C}_2, r_2 \rangle \in \llbracket \mathcal{P}_2 \mid \alpha \rrbracket, \text{ and} \\ \left. \mathcal{C} \subseteq \mathbf{in}(\mathcal{P}_1) \times \mathbf{out}(\mathcal{P}_2) \cup \mathbf{in}(\mathcal{P}_2) \times \mathbf{out}(\mathcal{P}_1) \text{ such that } (f_1 \parallel f_2) \models \mathcal{C} \right\}$$

4. If $\mathcal{M} = (\mathbf{let } X = \mathcal{P} \mathbf{ in } \mathcal{P}')$, then $\llbracket \mathcal{M} \mid \alpha \rrbracket = \llbracket \mathcal{P}' \mid \alpha \rrbracket$, provided two conditions:

(a) $\dim(X) \approx \dim(\mathcal{P})$,

(b) for every $f : \mathbf{in}(X) \cup \mathbf{out}(X) \rightarrow \mathbb{R}^+$, $B \subseteq \mathbf{in}(X) \cup \mathbf{out}(X)$, $\mathcal{C} \subseteq \mathbf{in}(X) \times \mathbf{out}(X)$, and $r \in \mathbb{R}^+$,

$$\langle f, B, \mathcal{C}, r \rangle \in \llbracket X \mid \alpha \rrbracket \quad \text{iff} \\ \text{there is } \langle g, B', \mathcal{C}', r \rangle \in \llbracket \mathcal{P} \mid \alpha \rrbracket \text{ such that } f \approx [g]_A, B \approx B', \text{ and } \mathcal{C} \approx \mathcal{C}',$$

where $A = \mathbf{in}(\mathcal{P}) \cup \mathbf{out}(\mathcal{P})$.

5. If $\mathcal{M} = \mathbf{bind}(\mathcal{P}, \langle a, b \rangle)$, then

$$\begin{aligned} \llbracket \mathcal{M} \mid \alpha \rrbracket &= \left\{ \langle f, B, \mathcal{C}, r \rangle \mid \langle f, B, \mathcal{C} \cup \{\langle b, a \rangle\}, r \rangle \in \llbracket \mathcal{P} \mid \alpha \rrbracket, B \cap \{a, b\} = \emptyset, \right. \\ &\quad \text{and for every } \langle g, B, \mathcal{C} \cup \{\langle b, a \rangle\}, s \rangle \in \llbracket \mathcal{P} \mid \alpha \rrbracket \\ &\quad \left. \text{if } [f]_B = [g]_B \text{ then } r \leq s \right\} \end{aligned}$$

It is now a straightforward proof by induction on the definition of \mathcal{N} to show that (IH) holds for every subexpression of \mathcal{N} and for \mathcal{N} itself. To conclude the proof, we simply observe that

$$\llbracket \mathcal{N} \mid \alpha \rrbracket = \left\{ \langle f, B, r \rangle \mid \langle f, B, \emptyset, r \rangle \in \llbracket \mathcal{N} \mid \alpha \rrbracket \right\}$$

which implies (IH) holds for the particular case when $\mathcal{C} = \emptyset$, which in turn implies the proposition. \square

11 A Relativized Typing System

Let α be an additive aggregate objective, *e.g.*, one of those mentioned in Section 10. Assume α is fixed and the same throughout this section, except in Example 57 where we instantiate α to the objectives HR, AU, and MD.

Let \mathcal{N} be a closed network specification. According to Section 8, if the judgment “ $\vdash \mathcal{N} : T$ ” is derivable using the rules in Figure 8 and T is a valid typing, then $\text{Poly}(T)$ is a set of feasible IO-flows in \mathcal{N} , *i.e.*, $\text{Poly}(T) \subseteq \llbracket \mathcal{N} \rrbracket$. And if T is principal, then in fact $\text{Poly}(T) = \llbracket \mathcal{N} \rrbracket$.

In this section, judgments are of the form “ $\vdash \mathcal{N} : (T, \Phi)$ ” and derived using the rules in Figure 12. 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 $\text{Poly}(T) \subseteq \llbracket \mathcal{N} \rrbracket$ (resp. $\text{Poly}(T) = \llbracket \mathcal{N} \rrbracket$), but now the auxiliary Φ is used to select members of $\text{Poly}(T)$ that minimize the objective α .

If this is going to work at all, then Φ should not be allowed to inspect the whole of \mathcal{N} . Instead, Φ should be defined inductively from the relativized typings for 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 $\llbracket \mathcal{N} \rrbracket$, as in Section 10. We now define it on $\llbracket \mathcal{N} \rrbracket$. For every $f \in \llbracket \mathcal{N} \rrbracket$, let:

$$\alpha(f) = \min \left\{ \alpha(f') \mid f' \in \llbracket \mathcal{N} \rrbracket \text{ and } f' \text{ extends } f \right\}.$$

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 $\text{Poly}(T) \subseteq \llbracket \mathcal{N} \rrbracket$. For economy of writing, let $\mathcal{F} = \text{Poly}(T)$. Relative to this T , we define the function Φ_T as follows:

$$\begin{aligned} \Phi_T &: \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathcal{P}(\mathcal{F} \times \mathbb{R}^+) \\ \Phi_T(B) &= \left\{ \langle f, r \rangle \mid f \in \mathcal{F}, r = \alpha(f), \text{ and for every } g \in \mathcal{F}, \text{ if } [f]_B = [g]_B, \text{ then } r \leq \alpha(g) \right\} \end{aligned}$$

where $B \in \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}})$. In words, $\Phi_T(B)$ selects f provided, among all members of $\mathcal{F} \subseteq \llbracket \mathcal{N} \rrbracket$ 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 $\llbracket \mathcal{N} \mid \alpha \rrbracket$ in Section 10. Note that, in this 5th clause, we define $\llbracket \mathcal{M} \mid \alpha \rrbracket$ without having to examine any of the internal properties of the immediate constituent part \mathcal{P} .

Definition 53 (*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 $\text{Poly}^*(T, \Phi)$ as a set of triples:

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

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

$$\text{Poly}^*(T, \Phi) = \left\{ \langle f, B, r \rangle \mid f \in \text{Poly}(T), B \subseteq \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}, r = \alpha(f), \right. \\ \left. \text{and for all } g \in \text{Poly}(T) \text{ if } [f]_B = [g]_B \text{ then } \alpha(f) \leq \alpha(g) \right\}$$

We say the relativized typing $(\mathcal{N} : (T, \Phi))$ is *valid* iff $\text{Poly}^*(T, \Phi) \subseteq \llbracket \mathcal{N} \mid \alpha \rrbracket$, and we say it is *principal* iff $\text{Poly}^*(T, \Phi) = \llbracket \mathcal{N} \mid \alpha \rrbracket$. Compare these definitions with their un-relativized counterparts at the beginning of Section 5.2. \square

A case of particular interest in the preceding definitions is when $B = \mathbf{A}_{\text{in}}$. Suppose $\langle f, \mathbf{A}_{\text{in}}, r \rangle \in \text{Poly}^*(T, \Phi)$. This means that, among all feasible flows g in \mathcal{N} agreeing with f on \mathbf{A}_{in} , f is α -optimal with $\alpha(f) = r$. More, in fact, for every α -optimal feasible flow g in \mathcal{N} agreeing with f on \mathbf{A}_{in} , we will have $\alpha(g) = r$, so that:

$$\left\{ \langle g, \mathbf{A}_{\text{in}}, r \rangle \mid [f]_{\mathbf{A}_{\text{in}}} = [g]_{\mathbf{A}_{\text{in}}} \text{ and } g \text{ is } \alpha\text{-optimal feasible flow in } \mathcal{N} \right\} \subseteq \text{Poly}^*(T, \Phi)$$

Note $\langle g, \mathbf{A}_{\text{in}}, r \rangle$ provides no information about the path taken by α -optimal feasible flow g *inside* \mathcal{N} . It only says the α -value of g is r , assuming g indeed uses an α -optimal path through \mathcal{N} . Nor does it say anything about the values assigned to output arcs by an α -optimal feasible flow g through \mathcal{N} , although we only know that

$$\sum f(\mathbf{A}_{\text{in}}) = \sum f(\mathbf{A}_{\text{out}}) = \sum g(\mathbf{A}_{\text{in}}) = \sum g(\mathbf{A}_{\text{out}})$$

because of flow conservation.

11.1 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 “ $\text{bind}((T, \Phi), \langle a, b \rangle)$ ”, whose definitions are based on clauses 3 and 5 in the inductive definition of $\llbracket \mathcal{N} \mid \alpha \rrbracket$ in Section 10.

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

- $T = (T_1 \parallel T_2)$, as defined at the beginning of Section 8.1,
- for every $B_1 \subseteq \text{in}(\mathcal{N}_1) \cup \text{out}(\mathcal{N}_1)$ and every $B_2 \subseteq \text{in}(\mathcal{N}_2) \cup \text{out}(\mathcal{N}_2)$:

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

Lemma 54. *If the relativized typings $(\mathcal{N}_1 : (T_1, \Phi_1))$ and $(\mathcal{N}_2 : (T_2, \Phi_2))$ are principal, resp. valid, then so is the relativized typing $(\mathcal{N}_1 \parallel \mathcal{N}_2) : ((T_1, \Phi_1) \parallel (T_2, \Phi_2))$ principal, resp. valid.*

Proof Sketch. Straightforward generalization of Lemma 42. \square

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

- $T^* = \text{bind}(T, \langle a, b \rangle)$, as defined in Section 8.1,
- for every $B \subseteq (\text{in}(\mathcal{P}) \cup \text{out}(\mathcal{P})) - \{a, b\}$:

$$\Phi^*(B) = \left\{ \langle [f]_B, r \rangle \mid \langle f, r \rangle \in \Phi(B \cup \{a, b\}), f(a) = f(b), \text{ and for all } \langle g, s \rangle \in \Phi(B \cup \{a, b\}) \right. \\ \left. \text{if } g(a) = g(b) \text{ and } [f]_B = [g]_B \text{ then } r \leq s \right\}$$

Lemma 55. *If the relativized typing $(\mathcal{P} : (T, \Phi))$ is principal, resp. valid, then so is the relativized typing $(\mathbf{bind}(\mathcal{P}, \langle a, b \rangle) : \text{bind}((T, \Phi), \langle a, b \rangle))$ principal, resp. valid.*

Proof Sketch. Straightforward generalization of Lemma 43. \square

11.2 Relativized Typing Rules

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

Figure 12: Relativized Typing Rules for Flow Networks.

The operations “ $(T_1, \Phi_1) \parallel (T_2, \Phi_2)$ ” and “ $\mathbf{bind}((T, \Phi), \langle a, b \rangle)$ ” are defined in Section 11.1.

Theorem 56 (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 12, 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} .

Proof Sketch. Similar to the proof of Theorem 45, now using Lemmas 54 and 55. □

Example 57 illustrates some of the preceding notions. Let T be a valid typing for network specification \mathcal{N} and $\mathcal{F} = \text{Poly}(T)$. If $\mathbf{A}_{\text{in}} = \mathbf{in}(\mathcal{N})$ and $\mathbf{A}_{\text{out}} = \mathbf{out}(\mathcal{N})$, we define the function φ_T according to:

$$\begin{aligned} \varphi_T & : \mathcal{P}(\mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}) \rightarrow \mathcal{P}(\mathcal{F}) \\ \varphi_T(B) & = \{ f \mid \langle f, r \rangle \in \Phi_T(B) \text{ for some } r \} \end{aligned}$$

i.e., $\varphi_T(B)$ is the same set as $\Phi_T(B)$ after throwing away the second entry of every pair in the latter. In words, $\varphi_T(B)$ selects all the α -optimal ones among the flows in \mathcal{F} that agree on B .

Recall a convenient shorthand representation of $f \in \langle\langle \mathcal{N} \rangle\rangle$ from Section 4. If $|\dim(\mathcal{N})| = m \geq 1$, we can write f as an m -tuple of non-negative reals:

$$f \text{ is represented by } \mathbf{r} = \langle r_1, \dots, r_m \rangle,$$

i.e., if $\dim(\mathcal{N}) = \langle a_1, \dots, a_m \rangle$, then $f(a_1) = r_1, \dots, f(a_m) = r_m$. More succinctly, we write $f(\dim(\mathcal{N})) = \mathbf{r}$. The zero flow through \mathcal{N} is represented by the m -tuple of zero's, $\langle 0, 0, \dots, 0 \rangle$, which can be extended to a feasible flow only if $L(a) = 0$ for every arc a . It is also convenient to define the “monus” function on real numbers as follows:

$$x \dot{-} y = \begin{cases} x - y & \text{if } x > y \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Example 57. Consider small network \mathcal{A} from Examples 13, 40, and 41. Here, $\dim(\mathcal{A}) = \langle a_1, a_2, a_3, a_4 \rangle$. For every arc a in this small network, the lower bound $L(a) = 0$. Let T be a valid typing for \mathcal{A} and write \mathcal{F} for $\text{Poly}(T)$. Hence, $\mathcal{F} \subseteq \langle\langle \mathcal{A} \rangle\rangle = \text{Poly}(T_{\mathcal{A}})$, where $T_{\mathcal{A}}$ is the principal typing of \mathcal{A} determined in Example 40.

The function on $\dim(\mathcal{A})$ represented by $\langle 0, 0, 0, 0 \rangle$ can be trivially extended to a feasible flow in \mathcal{A} which is HR-, AU-, and MD-optimal. Hence, if $B = \emptyset$, then:

$$\varphi_T^{\text{HR}}(\emptyset) = \varphi_T^{\text{AU}}(\emptyset) = \varphi_T^{\text{MD}}(\emptyset) = \{\langle 0, 0, 0, 0 \rangle\}$$

If $B = \{a_1, a_2, a_3, a_4\}$, then:

$$\varphi_T^{\text{HR}}(B) = \varphi_T^{\text{AU}}(B) = \varphi_T^{\text{MD}}(B) = \mathcal{F}$$

For the rest of this example, we omit the subscript “ T ”.

The more interesting cases to consider are for $\emptyset \neq B \neq \{a_1, a_2, a_3, a_4\}$. We consider $B = \{a_1\}$ and $B = \{a_1, a_2\}$ only, leaving the other cases of B such that $\emptyset \neq B \neq \{a_1, a_2, a_3, a_4\}$ to the reader. For $B = \{a_1\}$, and objectives HR and AU, it is readily checked that:

$$\begin{aligned} \varphi^{\text{HR}}(B) &= \mathcal{F} \cap \{ \langle r_1, 0, r_1 + s, -s \rangle \in \mathbb{R}^4 \mid -(r_1 \div 5) \leq s \leq 0 \} \\ \varphi^{\text{AU}}(B) &= \mathcal{F} \cap \{ \langle r_1, 0, r_1 + s, -s \rangle \in \mathbb{R}^4 \mid -\min\{r_1, 10\} \leq s \leq 0 \} \end{aligned}$$

For $B = \{a_1, a_2\}$, and objectives HR and AU again, by brute-force inspection:

$$\begin{aligned} \varphi^{\text{HR}}(B) &= \mathcal{F} \cap \{ \langle r_1, r_2, r_1 + s, r_2 - s \rangle \in \mathbb{R}^4 \mid -(r_1 \div 5) \leq s \leq r_2 \div 15 \} \\ \varphi^{\text{AU}}(B) &= \mathcal{F} \cap \{ \langle r_1, r_2, r_1 + s, r_2 - s \rangle \in \mathbb{R}^4 \mid -\min\{r_1, 10 - (r_2 \div 15)\} \leq s \leq r_2 \div 15 \} \end{aligned}$$

To see how we determined $\varphi^{\text{HR}}(B)$ with $B = \{a_1, a_2\}$, start with fixed values r_1 and r_2 in the intervals $[0, 15]$ and $[0, 25]$, such that $r_1 + r_2$ is also in the interval $[0, 30]$. Objective HR pushes incoming flow at arc a_1 through internal arc a_5 as much as possible before using arc a_8 , and incoming flow at arc a_2 through internal arc a_{11} as much as possible before using arc a_8 . By contrast, for the determination of $\varphi^{\text{AU}}(B)$, objective AU pushes incoming flow at arc a_1 through arc a_8 as much as possible before using arc a_5 , and incoming flow at arc a_2 through arc a_{11} as much as possible before using arc a_8 .

It is a little trickier to determine $\varphi^{\text{MD}}(B)$, because the function $\text{MD}(f)$ is non-linear in the arc flows $f(a)$. For $B = \{a_1\}$, a little examination shows:

$$\varphi^{\text{MD}}(B) = \mathcal{F} \cap \{ \langle r_1, 0, r_1 + s, -s \rangle \in \mathbb{R}^4 \mid -(r_1 - \frac{r_1 \div 5}{2}) \leq s \leq 0 \}$$

For $B = \{a_1, a_2\}$, the determination of $\varphi^{\text{MD}}(B)$ is more involved. By brute-force computation:

$$\begin{aligned} \varphi^{\text{MD}}(B) &= \\ &\mathcal{F} \cap \left\{ \langle r_1, r_2, r_1 + s, r_2 - s \rangle \in \mathbb{R}^4 \mid -\left[r_1 - \frac{r_1 \div (5 - (r_2 \div 5)/2)}{2} \right] \leq s \leq \frac{r_2 \div 5}{2} \quad \text{where } r_2 \leq 15 \right\} \\ &\cup \\ &\mathcal{F} \cap \left\{ \langle r_1, r_2, r_1 + s, r_2 - s \rangle \in \mathbb{R}^4 \mid -\left[\frac{r_1 \div ((r_2 - 5)/2 - 5)}{2} \right] \leq s \leq \frac{r_2 - 5}{2} \quad \text{where } r_2 \geq 15 \right\} \end{aligned}$$

where we consider separately the two cases, $0 \leq r_2 \leq 15$ and $15 \leq r_2 \leq 25$.

To see how we determined $\varphi^{\text{MD}}(B)$, start with a fixed r_1 in the interval $[0, 15]$, then choose r_2 in the interval $[0, 25]$ without violating $r_1 + r_2$ in the interval $[0, 30]$. Consider the quantity that must be minimized to achieve the objective MD:

$$\frac{1}{(5 - r_{1,1})^2} + \frac{1}{(10 - r_{1,2} - r_{2,2})^2} + \frac{1}{(15 - r_{2,1})^2}$$

where $r_{1,1}$ and $r_{1,2}$ are the portions of r_1 which use internal arcs a_5 (of capacity 5) and a_8 (of capacity 10), and $r_{2,1}$ and $r_{2,2}$ are the portions of r_2 which use internal arcs a_{11} (of capacity 15) and a_8 (of capacity 10). For $r_1 \leq 5$, objective MD tries to minimize $r_{1,1}$ and maximize $r_{1,2}$. By contrast, for $r_2 \leq 15$, objective MD tries to maximize $r_{2,1}$ and minimize $r_{2,2}$. When both $r_1 \geq 5$ and $r_2 \geq 15$, the two flows compete for use of arc a_8 . \square

References

- [BKLO09] Azer Bestavros, Assaf Kfoury, Andrei Lapets, and Michael Ocean. Safe Compositional Network Sketches: Tool and Use Cases. In *Proceedings of CRTS'09: The IEEE/RTSS Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Washington D.C., December 2009.
- [BKLO10] Azer Bestavros, Assaf Kfoury, Andrei Lapets, and Michael Ocean. Safe Compositional Network Sketches: The Formal Framework. In *Proceedings of HSCC'10: The 13th ACM International Conference on Hybrid Systems: Computation and Control (in conjunction with CPSWEEK)*, Stockholm, Sweden, April 2010.
- [BL06] Simon Balon and Guy Leduc. Dividing the Traffic Matrix to Approach Optimal Traffic Engineering. In *Proc. of 14th IEEE Int'l Conf. on Networks (ICON 2006)*, volume 2, pages 566–571, Singapore, September 2006.