2013-03-04

A typing theory for flow networks (part I)

Kfoury, Assaf

Computer Science Department, Boston University

A Typing Theory for Flow Networks (Part I)

Assaf Kfoury*
Boston University
Boston, Massachusetts, USA
kfoury@bu.edu

March 4, 2013

Abstract

A *flow network* $\mathcal{N}$ is a capacited finite directed graph, with multiple *sources* (or *input arcs* in the paper) and multiple *sinks* (or *output arcs*). A flow f in $\mathcal{N}$ is *feasible* if it satisfies the usual flow-conservation condition at every node and lower-bound/upper-bound capacity constraints at every arc. We develop an algebraic theory of feasible flows in such networks with several beneficial consequences.

We define and prove the correctness of an algorithm to infer, from a given flow network $\mathcal{N}$, an algebraic characterization T of *all* assignments f of values to the input and output arcs of $\mathcal{N}$ that can be extended to feasible flows g. We call such a characterization T a *principal* typing for $\mathcal{N}$, as there are other typings which are only *valid* for $\mathcal{N}$ because they define subsets of all input-output assignments f that can be extended to feasible flows g. A typing for $\mathcal{N}$ turns out to define a bounded convex polyhedral set (or polytope) in the n-dimensional vector space $\mathbb{R}^n$ where n is the total number of input and output arcs in $\mathcal{N}$. We then establish necessary and sufficient conditions for an arbitrary typing to be a principal typing for some flow network.

Based on these necessary and sufficient conditions, we define operations on typings that preserve their principality (to be typings for flow networks), and examine the implications for a typing theory of flow networks. These also support a divide-and-conquer approach to the design and analysis of flow-network algorithms.

*Partially supported by NSF award CCF-0820138

Contents

1 Introduction

The work we report herein is a little off the beaten track. So we briefly explain the background that led to it. It starts with the modeling and analysis of large systems that are assembled in an incremental and modular way, while preserving desirable safety properties and other system requirements.

Background and motivation. Many large-scale, safety-critical systems can be viewed as inter-connections of subsystems, or modules, each of which is a producer, consumer, or regulator of flows. These flows are characterized by a set of variables and a set of constraints thereof, reflecting inherent or assumed properties or rules governing how the modules operate and what constitutes safe operation. Our notion of flow encompasses streams of physical entities (e.g., vehicles on a road, fluid in a pipe), data objects (e.g., sensor network packets, video frames), or consumable resources (e.g., electric energy, compute cycles).

Traditionally, the design and implementation of such flow networks follows a bottom-up approach, enabling system designers to certify desirable safety invariants of the system as a whole: Properties of the full system depend on a complete determination of the underlying properties of all subsystems. For example, the development of real-time applications necessitates the use of real-time kernels so that timing properties at the application layer (top) can be established through knowledge and/or tweaking of much lower-level system details (bottom), such as worst-case execution or context-switching times [DL97, LBJ+95, Reg02], specific scheduling and power parameters [AMM01, PLS01, SLM98, Sta00], among many others.

While justifiable in some instances, this vertical approach does not lend itself well to emerging practices in the assembly of complex large-scale systems – namely, the integration of various subsystems into a whole by system integrators who may not possess the requisite expertise or knowledge of the internals of these subsystems [KBS04]. This latter alternative can be viewed as a horizontal and incremental approach to system design and implementation, which has significant merits with respect to scalability and modularity. However, it also poses a major challenge with respect to verifiable trustworthiness – namely, how to formally certify that the system as a whole will satisfy specific safety invariants and to determine formal conditions under which it will remain so, as it is augmented, modified, or subjected to local component failures.

Further elaboration on this background can be found in a series of companion reports and articles over the last three years [BKLO09, BKLO10, BK11, Kfo11b, Kfo11a, SBKL11].

Our proposed framework. In support of this broader agenda, we make a foray into a a well-established area of combinatorial optimization in this report – flow networks and their connections to linear programming – but from a different angle. Starting from a network $\mathcal{N}$ with multiple input arcs/sources and output arcs/sinks, we want to derive an algebraic characterization, which we call a *principal typing*, of *all* the safe flows from inputs to outputs in $\mathcal{N}$. (In this report, we identify the safe flows in $\mathcal{N}$ with the *feasible* flows in $\mathcal{N}$.)

More precisely, we want an algebraic characterization of all the assignments f of values to the sources and sinks of $\mathcal{N}$, herein called *input-output functions*, which can be extended to feasible flows g in $\mathcal{N}$. A principal typing T for network $\mathcal{N}$ is thus a certificate for $\mathcal{N}$'s good working order, which abstracts away $\mathcal{N}$'s internal details and incidental features, and encodes enough of its input-output properties for its safe placement as a component in a larger assembly.

Moreover, we want this characterization to satisfy a *modularity* property in the sense that, if $\mathcal{N}'$ is another network with principal typing T', and if we connect $\mathcal{N}$ and $\mathcal{N}'$ by linking some of their outputs to some of their inputs to obtain a new network, which we denote (only in this Introduction) $\mathcal{N} \oplus \mathcal{N}'$, then the principal typing of $\mathcal{N} \oplus \mathcal{N}'$ is obtained by direct (and relatively easy) algebraic operations on T and T' – without any need to re-examine the internal details of the two components $\mathcal{N}$ and $\mathcal{N}'$. Put differently, an analysis (to produce a principal typing) for the assembled network $\mathcal{N} \oplus \mathcal{N}'$ can be directly obtained from the analysis of $\mathcal{N}$ and the analysis of $\mathcal{N}'$.

And we want more. The desired characterization should also satisfy a *compositionality* property, in the sense that neither of the two principal typings T and T' depends on the other; that is, the analysis (to produce T) for $\mathcal{N}$ and the analysis (to produce T') for $\mathcal{N}'$ can be carried out independently of each other without knowledge that the two will be subsequently assembled together.[1]

A complementary view of the preceding is to start from an already defined typing T and use it as a specification, or system requirement, against which we design a network or test the behavior of an existing one. In this dual sense, we certify the safe behavior of an already-designed network $\mathcal{N}$, or we use T to guide the process of designing a network $\mathcal{N}$ satisfying T.

The first view of a *typing theory for flow networks* is one of analysis, and the second view is one of synthesis. Both are supported by our examination in this report; several examples will illustrate them.

Main results. What we call a flow-network *typing* T turns out to be (equivalent to) a bounded convex polyhedral set or *polytope*,[2] in the n-dimensional vector space $\mathbb{R}^n$ for some integer $n \geqslant 0$, subject to appropriately defined restrictions. If T is a typing for a flow network $\mathcal{N}$, then the dimension n is the total number of sources and sinks in $\mathcal{N}$. Our main results in this report, Part I, are:

- Theorem 26, in Section 5, certifies the correctness of an algorithm for inferring a principal typing T for an arbitrarily given flow network $\mathcal{N}$. It is *principal* for $\mathcal{N}$ because there are other typings that are only *valid* for $\mathcal{N}$ in that they define subsets of input-output functions f extendable to feasible flows g in $\mathcal{N}$.

In Part II of the report, our results are:

- Necessary and sufficient conditions for an arbitrary polytope to be a principal typing (for some flow network).
- Various operations on principal typings, which in turn support the modularity and compositionality described above.

All of our major results depend on ideas and methods from polyhedral analysis and convex optimization, for which we use standard references [BV09, Sch12], among many other good books on these topics.

Organization of the report. Section 2 introduces our formulation of flow networks, as capacited directed graphs with multiple inputs (source nodes) and outputs (sink nodes). Section 3 presents four relatively simple flow networks, carefully defined to exhibit various features of interest for the later examination. Section 4 precisely defines typings of flow networks as polytopes. Sections 2, 3, and 4, are essential background for the rest of the report, both Part I and Part II.

In Section 5 we present the main results in this Part I of the report, Theorem 26. Not to disrupt the focus of our presentation, we delay the proof to Appendix A. This result was already presented, and alternative proofs for it outlined, in earlier reports and conference proceedings [Kfo11a, Kfo11b, BK11, Kfo12]. The last of these earlier reports also includes more efficient algorithms to infer principal typings in particular cases of flow networks. The proof of Theorem 26 here is new and mostly algebraic, but it also involves a pre-processing of the given flow network's underlying directed graph in order to simplify its algebraic analysis.

[1] In the study of programming languages, there are syntax-directed, inductively defined, type systems that are modular but not compositional in our sense. A case in point is the so-called Hindley-Milner type system for ML-like functional languages, where the order matters in which types are inferred.

[2] The use of the word "polytope" in the literature is a little ambiguous. Throughout this report, we take a polytope to mean a polyhedral set which is both convex and bounded.

Acknowledgments. The work reported herein is a fraction of a collective effort involving several people, under the umbrella of the iBench Initiative at Boston University, co-directed by Azer Bestavros and myself. The website `https://sites.google.com/site/ibenchbu/` gives a list of current and past participants, and research activities. Several iBench participants were a captive audience for partial presentations of the included material, in several sessions over the last two years. Special thanks are due to them all.

2 Flow Networks

We take a *flow network* $\mathcal{N}$ as a pair $\mathcal{N} = (\mathbf{N}, \mathbf{A})$, where $\mathbf{N}$ is a finite set of nodes and $\mathbf{A}$ a finite set of directed arcs, with each arc connecting two distinct nodes (no self-loops). We write $\mathbb{R}$ and $\mathbb{R}_+$ for the sets of reals and non-negative reals, respectively. Such a flow network $\mathcal{N}$ is supplied with *capacity functions* on the arcs and a *producer/consumer* (or *supply/demand*) assignment on the nodes:

- Lower-bound capacity $\underline{\boldsymbol{c}} : \mathbf{A} \to \mathbb{R}_+$.
- Upper-bound capacity $\overline{\boldsymbol{c}} : \mathbf{A} \to \mathbb{R}_+$.
- Producer/consumer assignment $\boldsymbol{d} : \mathbf{N} \to \mathbb{R}$.

We assume $0 \leqslant \underline{\boldsymbol{c}}(a) \leqslant \overline{\boldsymbol{c}}(a)$ and $\overline{\boldsymbol{c}}(a) \neq 0$ for every $a \in \mathbf{A}$. If $\boldsymbol{d}(\nu) > 0$, then ν is a *producer* node; if $\boldsymbol{d}(\nu) < 0$, then ν is a *consumer* node; and if $\boldsymbol{d}(\nu) = 0$, then ν is a *transshipment* node. We also assume that the sum of supplies equals the sum of demands; the requirement (\$) below is therefore an invariant of all our transformations on flow networks:

(\$) $\quad \sum\{\, \boldsymbol{d}(\nu) \mid \nu \in \mathbf{N} \,\} = 0.$

We identify the two ends of an arc $a \in \mathbf{A}$ by writing $\mathit{tail}(a)$ and $\mathit{head}(a)$, with the understanding that flow moves from $\mathit{tail}(a)$ to $\mathit{head}(a)$. The set $\mathbf{A}$ of arcs is the disjoint union – written "$\uplus$" whenever we want to make it explicit – of three sets: the set $\mathbf{A}_\#$ of internal arcs, the set $\mathbf{A}_{\text{in}}$ of input arcs, and the set $\mathbf{A}_{\text{out}}$ of output arcs:

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

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. Since there are no self-loops, $\mathit{head}(a) \neq \mathit{tail}(a)$ for all $a \in \mathbf{A}_\#$.

We assume that $\mathbf{N} \neq \varnothing$, *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. With no loss of generality, we make the following *connectedness assumption*:

(\$\$) For every $a \in \mathbf{A}$ there is a directed path from an input arc to an output arc that visits a.

We do not assume $\mathcal{N}$ 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 "source node") and only one output arc (or "sink node").[3]

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

[3] Except for multiple input arcs/source nodes and multiple output arcs/sink nodes, the notion of a flow network as here defined is standard [AMO93]. The presence of multiple sources and multiple sinks is not incidental, but crucial to the way we will develop and use our typing theory.

We call a bounded, closed interval $[r, r']$ of real numbers (possibly negative) a *type*, and we call a *typing* a partial map T (possibly total) that assigns types to subsets of the input and output arcs. Formally, T is of the following form, where $\mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$:[4]

$$T \,:\, \mathscr{P}(\mathbf{A}_{\text{in,out}}) \,\to\, \mathcal{I}(\mathbb{R})$$

where $\mathscr{P}(\ )$ is the power-set operator, $\mathscr{P}(\mathbf{A}_{\text{in,out}}) = \{A \,|\, A \subseteq \mathbf{A}_{\text{in,out}}\}$, and $\mathcal{I}(\mathbb{R})$ is the set of bounded, closed intervals of reals:

$$\mathcal{I}(\mathbb{R}) \;=\; \left\{ [r, r'] \;\middle|\; r, r' \in \mathbb{R} \text{ and } r \leqslant r' \right\}.$$

As a function, T is not totally arbitrary and satisfies certain conditions, discussed in Section 4, which qualify it as a *network typing*.

Henceforth, we use the term "network" to mean "flow network" in the sense just defined.

2.1 Flow Conservation, Capacity Constraints, Type Satisfaction

Though obvious and entirely standard, we precisely state fundamental concepts to fix our notation for the rest of the report, in Definitions 1, 2, 3, and 4.

Definition 1 (*Flow Conservation*)**.** If A is a subset of arcs in $\mathcal{N}$ and f a flow in $\mathcal{N}$, we write $\sum f(A)$ to denote the sum of the flows assigned to all the arcs in A:

$$\sum f(A) \;=\; \sum\{\, f(a) \,|\, a \in A \,\}.$$

By convention, $\sum \varnothing = 0$. If $A = \{a_1, \ldots, a_p\}$ is the set of arcs entering a node ν, and $B = \{b_1, \ldots, b_q\}$ the set of arcs exiting ν, conservation of flow at ν is expressed by the linear equation:

$$(1) \quad \sum f(A) \;+\; \boldsymbol{d}(\nu) \;=\; \sum f(B).$$

There is one such equation E_ν for every node $\nu \in \mathbf{N}$ and $\mathscr{E} = \{\, E_\nu \mid \nu \in \mathbf{N} \,\}$ is the collection of all equations enforcing flow conservation in $\mathcal{N}$. □

Definition 2 (*Capacity Constraints*)**.** A flow f satisfies the capacity constraints at arc $a \in \mathbf{A}$ if:

$$(2) \quad \underline{\boldsymbol{c}}(a) \;\leqslant\; f(a) \;\leqslant\; \overline{\boldsymbol{c}}(a).$$

There are two such inequalities C_a for every arc $a \in \mathbf{A}$ and $\mathscr{C} = \{\, C_a \mid a \in \mathbf{A} \,\}$ is the collection of all inequalities enforcing capacity constraints in $\mathcal{N}$. □

Definition 3 (*Feasible Flows*)**.** A flow f is *feasible* iff two conditions:

- for every node $\nu \in \mathbf{N}$, 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. □

[4] Our notion of a "typing" as an assignment of types/intervals to 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. See [Jim96], and the longer [Jim95], for the origin of this different notion of "typing". These two reports also discuss the distinction between "modular" and "compositional", in the same sense we explained in Section 1, but now in the context of type inference for strongly-typed functional programs.

Definition 4 (*Type Satisfaction*)**.** Let $\mathcal{N}$ be a network with input/output arcs $\mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$, and let $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ be a typing over $\mathbf{A}_{\text{in,out}}$. We say the flow f *satisfies* T if, for every $A \in \mathscr{P}(\mathbf{A}_{\text{in,out}})$ for which $T(A)$ is defined with $T(A) = [r, r']$, it is the case that:

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

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

Notation 5. We use mostly the letter $\mathcal{N}$ possibly decorated (with a prime, double prime, tilde, etc.), and occasionally the letter $\mathcal{M}$, to range over the set of networks. To denote particular example networks, we exclusively use the letter $\mathcal{N}$ appropriately subscripted (as in $\mathcal{N}_1, \mathcal{N}_2$, etc.).

We use mostly the letters $\mathbf{A}_{\text{in}}$ and $\mathbf{A}_{\text{out}}$, and occasionally the letters $\mathbf{B}_{\text{in}}$ and $\mathbf{B}_{\text{out}}$, to denote the sets of input arcs and output arcs of flow networks. We also write $\mathbf{A}_{\text{in,out}}$ for the disjoint union $\mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$. Although this notation is a little ambiguous (not indicating which arc in $\mathbf{A}_{\text{in,out}}$ is an input and which is an output), the context will always disambiguate.

We use exclusively the letters $\mathscr{E}$ and $\mathscr{C}$ to range over sets of flow-conservation equations and sets of capacity-constraint inequalities. To denote particular examples of such sets, we appropriately subscript them (as in $\mathscr{E}_1$ and $\mathscr{C}_1$, $\mathscr{E}_2$ and $\mathscr{C}_2$, etc.).

We use mostly the letter T possibly decorated (with a prime, double prime, etc.), and occasionally the letters S and U, to range over typings. To denote particular typings, in all the examples, we exclusively use the letter T appropriately subscripted (as in T_1, T_2, etc.).

If $\mathcal{N}_i$ is a particular example network, with particular conservation equations $\mathscr{E}_j$ and constraint inequalities $\mathscr{C}_j$, and T_k is a particular typing inferred from $\mathscr{E}_j$ and $\mathscr{C}_j$ or is related to them in some way, we make the subscripts i, j, and k, all the same.

Sometimes, when a function definition may be open to some confusion, we use the symbol ":=" instead of "=" to emphasize that it is not an equality.

2.2 Constant Input/Output Arcs vs. Producer/Consumer Nodes

A producer node ν of, say, 5 units can be viewed as the *head* of an input arc a with $\underline{\boldsymbol{c}}(a) = \overline{\boldsymbol{c}}(a) = 5$. Similarly, a consumer node ν' of 10 units can be viewed as the *tail* of an input arc a' with $\underline{\boldsymbol{c}}(a') = \overline{\boldsymbol{c}}(a') = 10$. Such arcs a and a' are *constant* input/output arcs – they have to be always assigned the same value by a feasible flow.

Conversely, a constant input arc a with $\mathit{head}(a) = \nu$ and, say, $\underline{\boldsymbol{c}}(a) = \overline{\boldsymbol{c}}(a) = 15$ can be omitted after an update $\boldsymbol{d}(\nu) := \boldsymbol{d}(\nu) + 15$. Similarly, a constant output arc a' with $\mathit{tail}(a') = \nu'$ and, say, $\underline{\boldsymbol{c}}(a') = \overline{\boldsymbol{c}}(a') = 20$ can be omitted after an update $\boldsymbol{d}(\nu') := \boldsymbol{d}(\nu') - 20$.

Algebraically, therefore, in the flow conservation equation (1) in Definition 1 it does not matter whether we include a constant input/output arc on the left or make an appropriate update of $\boldsymbol{d}$ on the right.

Nevertheless, we make a distinction between constant input/output arcs and producer/consumer nodes. An input arc a, constant or not, is an arc whose free tail can be linked to the free head of some output arc; an output arc a', constant or not, is an arc whose free head can be linked to the free tail of some input arc.

Warning. The transformation from producer/consumer nodes to constant input/output arcs, and the converse transformation, must be done with care to preserve the invariant (\$) in the opening paragraph of Section 2.

3 Examples

Below are four examples of flow networks which we use repeatedly in later sections to illustrate different aspects of our examination. Unless explicitly specified to be a producer (resp., a consumer) indicated by a fat inward (resp., outward) arrow head, a node is assumed to be a transshipment node.

Example 6. The network $\mathcal{N}_1$ is on the left in Figure 1. We leave the lower bounds r, s, and t, on arcs a_2, a_3, and a_6, unspecified for the moment. Later, we choose r, s, and t, to illustrate other concepts. We use the arc names $\{a_1, \ldots, a_6\}$ as variables in the equations and inequalities below. There are 3 equations enforcing flow conservation in $\mathcal{N}_1$:

$$\mathscr{E}_1 \;=\; \{\; a_1 + a_5 = a_4\,, \quad a_2 + a_6 = a_5\,, \quad a_3 + a_6 = a_4 \;\}$$

There are 12 inequalities, with 2 for each of the 6 arcs, enforcing lower-bound and upper-bound constraints:

$$\mathscr{C}_1 \;=\; \{\; 0 \leqslant a_1 \leqslant 15\,, \quad r \leqslant a_2 \leqslant 50\,, \quad s \leqslant a_3 \leqslant 35\,, \quad 0 \leqslant a_4 \leqslant 40\,, \quad 0 \leqslant a_5 \leqslant 40\,, \quad t \leqslant a_6 \leqslant 40 \;\}$$

The network $\mathcal{N}_1'$ in the middle is obtained from $\mathcal{N}_1$ after turning one node into a producer and one node into a consumer, each of amount t. The invariant (\$) in Section 2 is preserved. The equations in $\mathscr{E}_1$ and inequalities in $\mathscr{C}_1$ are adjusted accordingly:

$$\mathscr{E}_1' \;=\; \{\; a_1 + a_5 = a_4\,, \quad a_2 + a_6 + t = a_5\,, \quad a_3 + a_6 + t = a_4 \;\}$$

$$\mathscr{C}_1' \;=\; \{\; 0 \leqslant a_1 \leqslant 15\,, \quad r \leqslant a_2 \leqslant 50\,, \quad s \leqslant a_3 \leqslant 35\,, \quad 0 \leqslant a_4 \leqslant 40\,, \quad 0 \leqslant a_5 \leqslant 40\,, \quad 0 \leqslant a_6 \leqslant 40 - t \;\}$$

In $\mathcal{N}_1'$ and in contrast to $\mathcal{N}_1$, because the lower-bound capacity $\underline{c}(a_6) = 0$, feasible flows from inputs to outputs can be restricted to move along one of two acyclic paths, $a_1\, a_4\, a_3$ and $a_2\, a_5\, a_4\, a_3$.

The network $\mathcal{N}_1''$ on the right in Figure 1 is obtained from $\mathcal{N}_1'$ by changing the producer/consumer nodes to transshipment nodes and introducing corresponding constant input/output arcs – and then merging the constant input arc with a_2 and the constant output arc with a_3. Again, the invariant (\$) is preserved. □

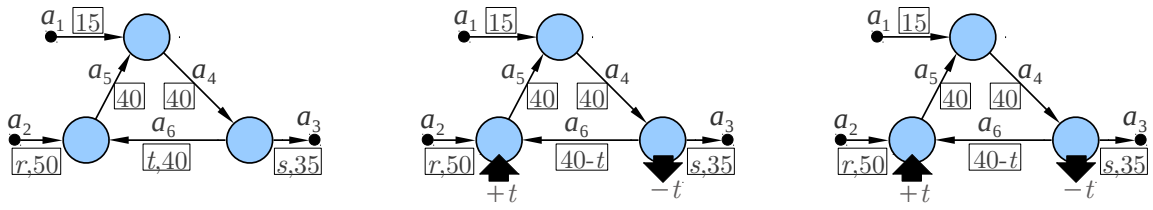

Figure 1: Network $\mathcal{N}_1$ on the left. Network $\mathcal{N}_1'$ in the middle, obtained from $\mathcal{N}_1$ by introducing producer/consumer nodes, each of amount t, indicated by heavy arrow heads. Network $\mathcal{N}_1''$ on the right, obtained from $\mathcal{N}_1'$ by introducing constant input/output arcs in place of producer/consumer nodes. If only one capacity is shown for an arc, it is an upper bound; all omitted lower bounds are 0. The lower bounds r, s and t are specified in follow-up examples.

Example 7. Network $\mathcal{N}_2$ is shown in Figure 2. We first list the collection $\mathscr{E}_2$ of 8 equations enforcing flow conservation in $\mathcal{N}_2$:

$$\begin{aligned}\mathscr{E}_2 \;=\; \Big\{\; & a_1 = a_6 + a_7\,, \quad a_2 = a_8 + a_9\,, \quad a_3 + a_{18} = a_{10} + a_{11}\,, \\ & a_6 + a_8 = a_{12}\,, \quad a_7 + a_9 + a_{10} = a_{13} + a_{14}\,, \quad a_{14} + a_{15} = a_{17} + a_{18}\,, \\ & a_{12} + a_{13} + a_{17} = a_4 + a_{16}\,, \quad a_{11} + a_{16} = a_5 + a_{15} \;\Big\}\end{aligned}$$

We use the arc names $\{a_1, \ldots, a_{18}\}$ as variables in the preceding equations, and again in the collection $\mathscr{C}_2$ of $2 \cdot 18 = 36$ inequalities enforcing lower-bound and upper-bound constraints:

$$\begin{aligned}\mathscr{C}_2 \;=\; \Big\{\; & 2 \leqslant a_1 \leqslant 15\,, \quad 0 \leqslant a_2 \leqslant 20\,, \quad 4 \leqslant a_3 \leqslant 25\,, \quad 3 \leqslant a_4 \leqslant 8\,, \quad 4 \leqslant a_5 \leqslant 15\,, \quad 0 \leqslant a_6 \leqslant 5\,, \\ & 0 \leqslant a_7 \leqslant 5\,, \quad 0 \leqslant a_8 \leqslant 2\,, \quad 0 \leqslant a_9 \leqslant 10\,, \quad 0 \leqslant a_{10} \leqslant 10\,, \quad 0 \leqslant a_{11} \leqslant 4\,, \quad 0 \leqslant a_{12} \leqslant 5\,, \\ & 0 \leqslant a_{13} \leqslant 3\,, \quad 0 \leqslant a_{14} \leqslant 2\,, \quad 0 \leqslant a_{15} \leqslant 3\,, \quad 0 \leqslant a_{16} \leqslant 10\,, \quad 0 \leqslant a_{17} \leqslant 7\,, \quad 0 \leqslant a_{18} \leqslant 6 \;\Big\}\end{aligned}$$

We can compute the values of a maximum feasible flow and a minimum feasible flow using linear programming, *e.g.*, the *network simplex method* [Cun76, Cun79]. Alternatively, we can use standard algorithms on capacited graphs, *e.g.*, the *min-cut/max-flow* theorem and the *max-cut/min-flow* theorem (Theorem 7.2 in [Sch86], Theorem 4.2 in [Sch12]).

The upper-bound capacity of a min-cut Φ is the value of a feasible max flow in $\mathcal{N}_2$, and the lower-bound capacity of a max-cut Ψ is the value of a feasible min flow in $\mathcal{N}_2$. (In general, there are more than one of each, but not in the network $\mathcal{N}_2$.) The upper-bound capacity 14 of the min-cut Φ is obtained according to the formula:

$$\sum\{\text{upper-bounds of forward arcs in } \Phi\} - \sum\{\text{lower-bounds of backward arcs in } \Phi\} = 14$$

and the lower-bound capacity 4 of the max-cut Ψ is obtained according to the formula:

$$\sum\{\text{lower-bounds of forward arcs in } \Psi\} - \sum\{\text{upper-bounds of backward arcs in } \Psi\} = 7$$

Φ and Ψ are shown in Figure 3. Hence, the value of any feasible flow in $\mathcal{N}_2$ will be in the interval $[7, 14]$ and the types assigned by a typing T will have to enforce these limits, among other things. □

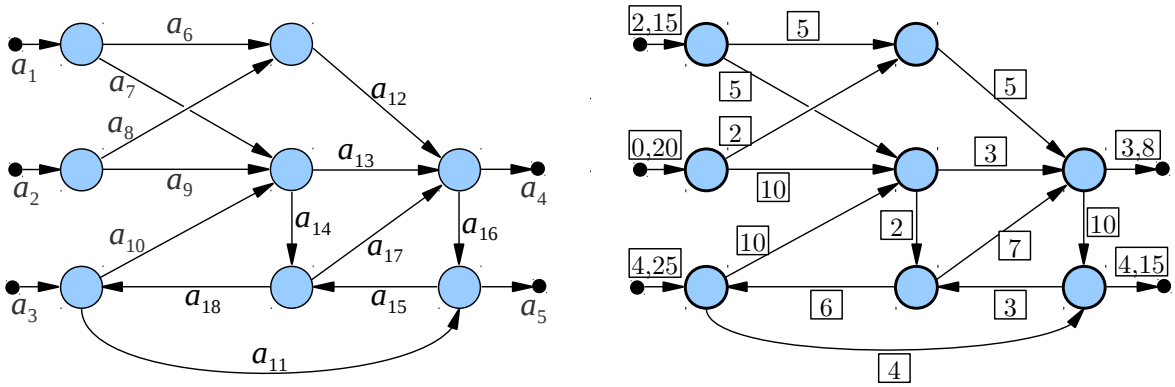

Figure 2: Network $\mathcal{N}_2$ with its named arcs is on the left, and with its lower-bound and upper-bound capacities on the right. If only one capacity is shown, it is an upper bound; omitted lower bounds are 0.

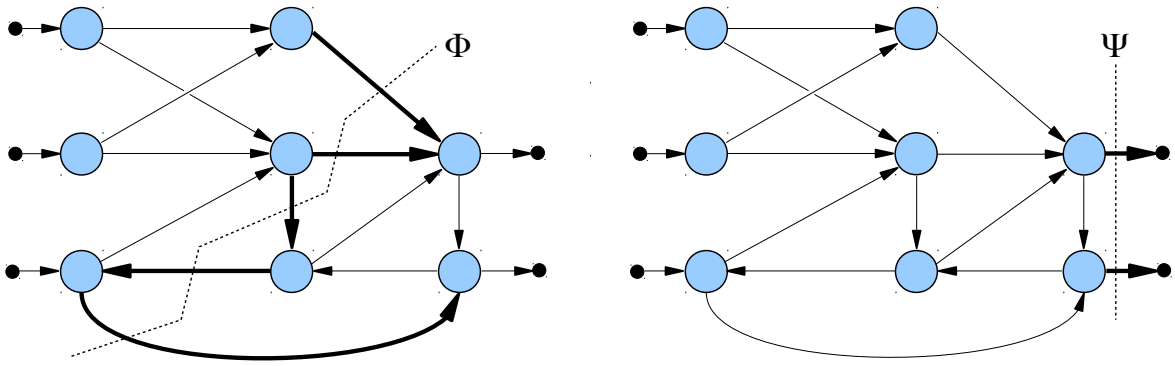

Figure 3: The min cut $\Phi = \{a_{11}, a_{12}, a_{13}, a_{14}, a_{18}\}$ and the max cut $\Psi = \{a_4, a_5\}$ in $\mathcal{N}_2$.

Example 8. Network $\mathcal{N}_3$ is shown on the left in Figure 4. There are 6 equations in $\mathscr{E}_3$ enforcing flow conservation, one for each node in $\mathcal{N}_3$, and $2 \cdot 11 = 22$ inequalities in $\mathscr{C}_3$ enforcing lower-bound and upper-bound constraints, two for each arc in $\mathcal{N}_3$. We omit the straightforward $\mathscr{E}_3$ and $\mathscr{C}_3$.

In Figure 4, all omitted lower-bound capacities are 0 and all omitted upper-bound capacities are K. K is an unspecified "very large number".

By easy inspection, a minimum flow in $\mathcal{N}_3$ pushes 0 units through, and a maximum flow in $\mathcal{N}_3$ pushes 30 units. The value of every feasible flow in $\mathcal{N}_3$ will therefore be in the interval $[0, 30]$.

An appropriate typing for $\mathcal{N}_3$ will specify a permissible interval at each of the outer arcs $\{a_1, a_2, a_3, a_4\}$ so that the total flow pushed through $\mathcal{N}_3$ remains within the interval $[0, 30]$. □

Example 9. Network $\mathcal{N}_4$ is shown on the right in Figure 4. There are 8 equations in $\mathscr{E}_4$ enforcing flow conservation, one for each node in $\mathcal{N}_4$, and $2 \cdot 16 = 32$ inequalities in $\mathscr{C}_4$ enforcing lower-bound and upper-bound constraints, two for each arc in $\mathcal{N}_4$. We omit the straightforward $\mathscr{E}_4$ and $\mathscr{C}_4$.

By inspection, a minimum flow in $\mathcal{N}_4$ pushes 0 units through, and a maximum flow in $\mathcal{N}_4$ pushes 30 units. The value of all feasible flows in $\mathcal{N}_4$ will therefore be in the interval $[0, 30]$, the same as for $\mathcal{N}_3$ in Example 8.

However, as we will note when we re-visit $\mathcal{N}_3$ and $\mathcal{N}_4$ in Examples 24 and 25, an appropriate typing for the first will not be necessarily appropriate for the second, nor vice-versa. This will imply, among other things, there are maximum-value flows in $\mathcal{N}_3$ assigning values to the outer arcs $\{a_1, a_2, a_3, a_4\}$ which are different from those implied by a maximum-value flow in $\mathcal{N}_4$. □

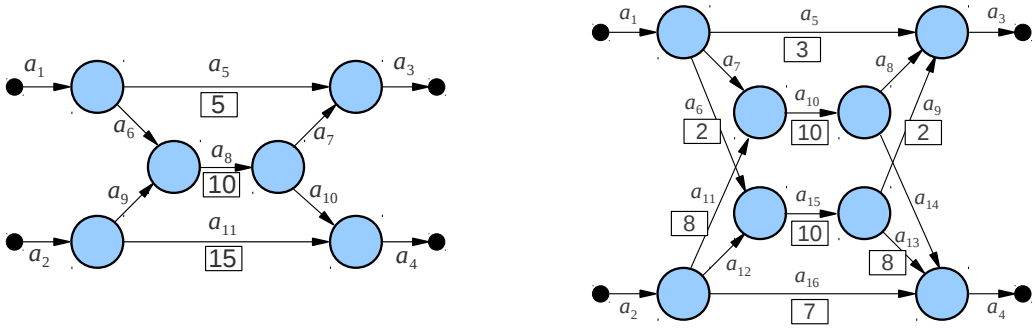

Figure 4: Network $\mathcal{N}_3$ (on the left) in Example 8 and network $\mathcal{N}_4$ (on the right) in Example 9. All missing capacities are the trivial lower bound 0 and the trivial upper bound K (a "very large number"). All feasible flows in both $\mathcal{N}_3$ and $\mathcal{N}_4$ have values in the interval/type $[0, 30]$.

4 Flow-Network Typings

Let $\mathbf{A} = \mathbf{A}_{\#} \uplus \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$ be the set of arcs in a network, with $\mathbf{A}_{\text{in}} = \{a_1, \ldots, a_m\}$, $\mathbf{A}_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\}$, and $\mathbf{A}_{\#} = \{a_{m+n+1}, \ldots, a_{m+n+p}\}$, where $m, n \geqslant 1$ and $p \geqslant 0$. As before, $\mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$ and we call a partial map T of the form:[5]

$$T \,:\, \mathscr{P}(\mathbf{A}_{\text{in,out}}) \,\to\, \mathcal{I}(\mathbb{R})$$

a *typing* over $\mathbf{A}_{\text{in,out}}$. A typing T over $\mathbf{A}_{\text{in,out}}$ defines a convex *polyhedral set*, which we denote $\mathsf{Poly}(T)$, in the Euclidean hyperspace $\mathbb{R}^{m+n}$, as we explain next. We think of the $m + n$ arcs in $\mathbf{A}_{\text{in,out}}$ as the dimensions of the space $\mathbb{R}^{m+n}$, and we thus use the arc names as variables to which we assign values in $\mathbb{R}$. $\mathsf{Poly}(T)$ is the

[5]The notation "$\mathbf{A}_{\text{in,out}}$" is ambiguous, because it does not distinguish between input and output arcs. We use it nonetheless for succintness. The context will always make clear which members of $\mathbf{A}_{\text{in,out}}$ are input arcs and which are output arcs.

intersection of at most $2\cdot(2^{m+n}-1)$ halfspaces, because there are $(2^{m+n}-1)$ non-empty subsets in $\mathscr{P}(\mathbf{A}_{\text{in,out}})$ and each induces two inequalities. Let $\varnothing \neq A \subseteq \mathbf{A}_{\text{in,out}}$ with:

$$A \cap \mathbf{A}_{\text{in}} = \{a'_1, \ldots, a'_k\} \quad \text{and} \quad A \cap \mathbf{A}_{\text{out}} = \{a'_{k+1}, \ldots, a'_\ell\}.$$

Suppose $T(A)$ is defined and let $T(A) = [r, r']$. Corresponding to A, there are two linear inequalities in the variables $\{a'_1, \ldots, a'_\ell\}$, denoted $T^{\min}_{\geqslant}(A)$ and $T^{\max}_{\leqslant}(A)$:

$$\begin{array}{lll} T^{\min}_{\geqslant}(A): & a'_1 + \cdots + a'_k - a'_{k+1} - \cdots - a'_\ell \geqslant r & \text{or, more succintly, } \sum(A \cap \mathbf{A}_{\text{in}}) - \sum(A \cap \mathbf{A}_{\text{out}}) \geqslant r \\ T^{\max}_{\leqslant}(A): & a'_1 + \cdots + a'_k - a'_{k+1} - \cdots - a'_\ell \leqslant r' & \text{or, more succintly, } \sum(A \cap \mathbf{A}_{\text{in}}) - \sum(A \cap \mathbf{A}_{\text{out}}) \leqslant r' \end{array} \tag{4}$$

and, therefore, two halfspaces $\mathsf{Half}(T^{\min}_{\geqslant}(A))$ and $\mathsf{Half}(T^{\max}_{\leqslant}(A))$ in $\mathbb{R}^{m+n}$ defined by:

$$\begin{aligned} \mathsf{Half}(T^{\min}_{\geqslant}(A)) &:= \{\, \boldsymbol{r} \in \mathbb{R}^{m+n} \mid \boldsymbol{r} \text{ satisfies } T^{\min}_{\geqslant}(A) \,\}, \\ \mathsf{Half}(T^{\max}_{\leqslant}(A)) &:= \{\, \boldsymbol{r} \in \mathbb{R}^{m+n} \mid \boldsymbol{r} \text{ satisfies } T^{\max}_{\leqslant}(A) \,\}. \end{aligned} \tag{5}$$

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

$$\boxed{\mathsf{Poly}(T) := \bigcap \left\{ \mathsf{Half}(T^{\min}_{\geqslant}(A)) \cap \mathsf{Half}(T^{\max}_{\leqslant}(A)) \;\middle|\; \varnothing \neq A \subseteq \mathbf{A}_{\text{in,out}} \text{ and } T(A) \text{ is defined} \right\}}$$

Generally, many of the inequalities induced by the typing T will be redundant, and the induced $\mathsf{Poly}(T)$ will be defined by far fewer than $2\cdot(2^{m+n}-1)$ halfspaces. For later reference, we give the name $\mathsf{Constraints}(T)$ to the set of all inequalities/constraints that define $\mathsf{Poly}(T)$:

$$\begin{aligned} \mathsf{Constraints}(T) := {} & \{\, T^{\min}_{\geqslant}(A) \mid \varnothing \neq A \subseteq \mathbf{A}_{\text{in,out}} \text{ and } T(A) \text{ is defined} \,\} \\ & \cup\ \{\, T^{\max}_{\leqslant}(A) \mid \varnothing \neq A \subseteq \mathbf{A}_{\text{in,out}} \text{ and } T(A) \text{ is defined} \,\}. \end{aligned} \tag{6}$$

Restriction 10. We agree that, in order for $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ to be a typing, three requirements must be satisfied:

1. $T(\varnothing) = T(\mathbf{A}_{\text{in,out}}) = [0,0] = \{0\}$. Informally, $T(\mathbf{A}_{\text{in,out}}) = \{0\}$ expresses *global flow conservation*: The total amount entering a network must equal the total amount exiting it – under the assumption that the sum of internal supplies is equal to the sum of internal demands – or after turning all producer/consumer nodes into constant input/output arcs, which is always possible to assume by the discussion in Section 2.2.

2. $\mathsf{Poly}(T)$ must be a *bounded* subspace of $\mathbb{R}^{m+n}$ and therefore a convex *polytope*, rather than just a convex *polyhedral set*. This means that for every $1 \leqslant i \leqslant m+n$, there is an bounded interval $[s, s']$, such that for every $\langle r_1, \ldots, r_i, \ldots, r_{m+n}\rangle \in \mathsf{Poly}(T)$, it must be that $s \leqslant r_i \leqslant s'$. This is a mild restriction, obviating the need to deal separately with cases of unboundedly large flows.

3. $\mathsf{Poly}(T)$ is entirely contained within the *first orthant* of the hyperspace $\mathbb{R}^{m+n}$, *i.e.*, the subspace $(\mathbb{R}_+)^{m+n}$. This means that if $\langle r_1, \ldots, r_{m+n}\rangle \in \mathsf{Poly}(T)$ then every component r_i is non-negative, corresponding to the fact that if an IO function $f : \mathbf{A}_{\text{in,out}} \to \mathbb{R}_+$ satisfies T, then every entry in $\langle f(a_1), \ldots, f(a_{m+n})\rangle$ is non-negative.

We want to characterize the typings T that are inhabited; specifically, we want to formulate necessary and sufficient conditions (preferably algebraic) that select, among all typings, those that are *tight* and *principal* typings (of networks) – which we define in the two next subsections.

Definition 11 (*Input-Output Functions*)**.** Let $\mathbf{A} = \mathbf{A}_{\#} \uplus \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$ be the set of arcs in a network $\mathcal{N}$, where $\mathbf{A}_{\text{in}} = \{a_1, \ldots, a_m\}$ and $\mathbf{A}_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\}$. We call a function $f : \mathbf{A}_{\text{in,out}} \to \mathbb{R}_+$ an *input-output function* or, or more briefly, an *IO function* for $\mathcal{N}$, where $\mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}}$ as before.

If $f' : \mathbf{A} \to \mathbb{R}_+$ is a flow in the network $\mathcal{N}$, then the restriction of f' to $\mathbf{A}_{\text{in,out}}$, denoted $\left[f'\right]_{\mathbf{A}_{\text{in,out}}}$, is an IO function. We say that an IO function $f : \mathbf{A}_{\text{in,out}} \to \mathbb{R}_+$ is *feasible* if there is a feasible flow $f' : \mathbf{A} \to \mathbb{R}_+$ such that $f = \left[f'\right]_{\mathbf{A}_{\text{in,out}}}$.

A typing $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ for $\mathcal{N}$ is defined independently of the internal arcs $\mathbf{A}_{\#}$. Hence, the notion of *satisfaction* of T by a flow f' as in Definition 4 directly applies, with no change, to an IO function f. □

Proposition 12 (Typing Satisfaction for Input-Output Functions)**.** *Let $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ be a typing and let $f : \mathbf{A}_{\text{in,out}} \to \mathbb{R}_+$ be an IO function. Then f satisfies T iff*

$$\langle f(a_1), \ldots, f(a_{m+n}) \rangle \in \mathsf{Poly}(T)$$

i.e., the point determined by f in the $(m+n)$-dimensional hyperspace is inside $\mathsf{Poly}(T)$. By a slight abuse of notation, we may write "$f \in \mathsf{Poly}(T)$" to indicate that f satisfies this condition.

Proof. This readily follows from Definition 4 and the notions introduced earlier in this section. □

4.1 Uniqueness and Redundancy in Typings

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

If T and U are typings over $\mathbf{A}_{\text{in,out}}$, we write $T \equiv U$ whenever $\mathsf{Poly}(T) = \mathsf{Poly}(U)$, in which case we say that T and U are *equivalent*.

Definition 13 (*Tight Typings*)**.** Let T be a typing over $\mathbf{A}_{\text{in,out}}$. T is *tight* if, for every $A \in \mathscr{P}(\mathbf{A}_{\text{in,out}})$ for which $T(A)$ is defined and for every $r \in T(A)$, there is an IO function $f \in \mathsf{Poly}(T)$ such that

$$r = \sum f(A \cap \mathbf{A}_{\text{in}}) - \sum f(A \cap \mathbf{A}_{\text{out}}).$$

Informally, T is tight if none of the intervals/types assigned by T to members of $\mathscr{P}(\mathbf{A}_{\text{in,out}})$ contains redundant information.[6] □

Let T be a typing over $\mathbf{A}_{\text{in,out}}$. If $T(A)$ is defined for $A \subseteq \mathbf{A}_{\text{in,out}}$, with $T(A) = [r_1, r_2]$ for some $r_1 \leqslant r_2$, we write $T^{\min}(A)$ and $T^{\max}(A)$ to denote the endpoints of $T(A)$:

$$T^{\min}(A) = r_1 \quad \text{and} \quad T^{\max}(A) = r_2.$$

The following is sometimes an easier-to-use characterization of tight typings.

Proposition 14 (Tightness Defined Differently)**.** *Let $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ be a typing. T is tight iff, for every $A \subseteq \mathbf{A}_{\text{in,out}}$ for which $T(A)$ is defined, there are $f_1, f_2 \in \mathsf{Poly}(T)$ such that:*

$$T^{\min}(A) = \sum f_1(A \cap \mathbf{A}_{\text{in}}) - \sum f_1(A \cap \mathbf{A}_{\text{out}}),$$
$$T^{\max}(A) = \sum f_2(A \cap \mathbf{A}_{\text{in}}) - \sum f_2(A \cap \mathbf{A}_{\text{out}}).$$

[6] There are different equivalent ways of defining "tightness". Let $\mathsf{Constraints}(T)$ be the set of inequalities induced by T, as in (6) above. Let $T^{\min}_{=}(A)$ and $T^{\max}_{=}(A)$ be the equations obtained by turning "$\geqslant$" and "$\leqslant$" into "=" in the inequalities $T^{\min}_{\geqslant}(A)$ and $T^{\max}_{\leqslant}(A)$ in $\mathsf{Constraints}(T)$. Using the terminology of [Sch86], pp 327, we say $T^{\min}_{=}(A)$ is *active* for $\mathsf{Poly}(T)$ if $T^{\min}_{=}(A)$ defines a face of $\mathsf{Poly}(T)$, and similarly for $T^{\max}_{=}(A)$. We can then say that T is tight if, for every $T^{\min}_{\geqslant}(A)$ and every $T^{\max}_{\leqslant}(A)$, the corresponding $T^{\min}_{=}(A)$ and $T^{\max}_{=}(A)$ are active for $\mathsf{Poly}(T)$.

Proof. The left-to-right implication follows immediately from Definition 13. The right-to-left implication is a staightforward consequence of the linearity of the constraints that define T. □

We state two additional easy results about tight typings, which we use in later sections.

Proposition 15 (Tightness Inherited Downward)**.** *Let $T, U : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ be typings such that $T \subseteq U$, i.e., U extends T. If U is tight, then so is T tight.*

Proof. Two preliminary observations, both following from $T \subseteq U$:

1. For every $A \in \mathscr{P}(\mathbf{A}_{\text{in,out}})$, if $T(A)$ is defined, so is $U(A)$ defined with $T(A) = U(A)$.
2. $\mathsf{Poly}(T) \supseteq \mathsf{Poly}(U)$, because $\mathsf{Constraints}(T) \subseteq \mathsf{Constraints}(U)$.

We need to show that for every $A \in \mathscr{P}(\mathbf{A}_{\text{in,out}})$ for which $T(A)$ is defined and for every $r \in T(A)$, there is an IO function $f \in \mathsf{Poly}(T)$ such that the following equation holds:

$$r = \sum f(A \cap \mathbf{A}_{\text{in}}) - \sum f(A \cap \mathbf{A}_{\text{out}}).$$

If $T(A)$ is defined, then $U(A)$ is defined, and if $r \in T(A)$, then $r \in U(A)$, by observation 1. Because U is tight, there is $f \in \mathsf{Poly}(U)$ such that the preceding equation holds. But $f \in \mathsf{Poly}(U)$ implies $f \in \mathsf{Poly}(T)$, by observation 2, from which the desired conclusion follows. □

Proposition 16 (Every Typing Is Equivalent to a Tight Typing)**.** *There is an algorithm* $\mathsf{Tight}()$ *which, given a typing T as input, always terminates and returns an equivalent tight (and total) typing* $\mathsf{Tight}(T)$.[7]

Proof. Starting from the given typing $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$, we first determine the set of linear inequalities $\mathsf{Constraints}(T)$ that defines $\mathsf{Poly}(T)$, as given in (6) above. We compute a total and tight typing $T' : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ by assigning an appropriate interval/type $T'(A)$ to every $A \in \mathscr{P}(\mathbf{A}_{\text{in,out}})$ as follows. For such a set A of input/output arcs, let θ_A be the objective function: $\theta_A = \sum A \cap \mathbf{A}_{\text{in}} - \sum A \cap \mathbf{A}_{\text{out}}$. Relative to $\mathsf{Constraints}(T)$, using standard procedures of linear programming, we minimize and maximize θ_A to obtain two values r_1 and r_2, respectively. The desired type $T'(A)$ is $[r_1, r_2]$ and the desired $\mathsf{Tight}(T)$ is T'. □

4.2 Valid Typings and Principal Typings

Let $\mathcal{N}$ be a network with input arcs $\mathbf{A}_{\text{in}}$ and output arcs $\mathbf{A}_{\text{out}}$. Let $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ be a typing over $\mathbf{A}_{\text{in,out}}$ and $\mathcal{N}$ a network. We say T is a *valid typing* for $\mathcal{N}$, sometimes denoted $(\mathcal{N} : T)$, if it is sound in the following sense:

(soundness) Every IO function $f : \mathbf{A}_{\text{in,out}} \to \mathbb{R}_+$ satisfying T can be extended to a feasible flow $g : \mathbf{A} \to \mathbb{R}_+$.

For a network $\mathcal{N}$, we say the typing $(\mathcal{N} : T)$ is a *principal typing* if it is both sound *and* complete:

(completeness) Every feasible flow $g : \mathbf{A} \to \mathbb{R}_+$ satisfies T.

Sometimes we say "a typing T is valid" (or "principal"), by which we mean that T is valid (or principal) for some network $\mathcal{N}$.

A useful notion in type theories is *subtyping*. Let T and U be valid typings over the same set of input/output arcs $\mathbf{A}_{\text{in,out}}$, *i.e.*, $T, U : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$. If T is a *subtyping* of U, in symbols $T <: U$, then any object for which T is a valid typing can be safely used in a context where an object with typing U is expected:

(subtyping) $\quad T <: U \quad$ iff $\quad \mathsf{Poly}(U) \subseteq \mathsf{Poly}(T)$.

[7] We leave out complexity and efficiency related to algorithm $\mathsf{Tight}()$ in this report, though they are important for follow-up work.

Note we require that both T and U be valid in order that "<:" work as expected, *i.e.*, if $\mathsf{Poly}(U) \subseteq \mathsf{Poly}(T)$ and T or U is not valid, then it is not necessarily that $T <: U$.

Our subtyping relation is contravariant w.r.t. the subset relation, *i.e.*, the supertyping U is more restrictive than the subtyping T. From the definition of equivalence between typings in Section 4.1, if both $T <: U$ and $U <: T$, then $T \equiv U$, naturally enough.[8]

Remark 17. The notion of subtyping is fundamental in typing theories for strongly-typed and object-oriented programming languages. It defines formal conditions for the safe substitution of a component for another, *i.e.*, without harming the behavior of the larger assembly in which components are inserted.

When components behave non-deterministically, the same substitution may or may not be safe, dependent on whether non-determinism is *angelic* or *demonic*. More specifically for our situation, an assignment of values to a network $\mathcal{N}$'s input arcs, $f_{\text{in}} : \mathbf{A}_{\text{in}} \to \mathbb{R}_+$, does not uniquely determine an assignment of values to $\mathcal{N}$'s output arcs, $f_{\text{out}} : \mathbf{A}_{\text{out}} \to \mathbb{R}_+$, nor vice-versa. Flow along internal arcs of $\mathcal{N}$ is non-deterministic. If we start from an assignment f_{in} at the input, angelic non-determinism tries to make "good" choices in order to preserve safety, whereas demonic non-determinism is adversarial and tries to make "bad" choices in order to disrupt safety.

Our notion of subtyping, and with it the notion of safe substitution, works as expected if non-determinism is angelic. □

Proposition 18 (Principal Typings Are Subtypings of Valid Typings)**.** *If $(\mathcal{N} : T)$ is a principal typing, and $(\mathcal{N} : U)$ a valid typing for the same $\mathcal{N}$, then $T <: U$.*

Proof. Given an arbitrary $f : \mathbf{A}_{\text{in,out}} \to \mathbb{R}_+$, it suffices to show that if f satisfies T_2, then f satisfies T, *i.e.*, any point in $\mathsf{Poly}(U)$ is also in $\mathsf{Poly}(T)$. If f satisfies U, then f can be extended to a feasible flow g. Because T is principal, g satisfies T. This implies that the restriction of g to $\mathbf{A}_{\text{in,out}}$, which is exactly f, satisfies T. □

Any two principal typings T and U for the same network are not necessarily identical, but they always denote the same polytope, as formally stated in the next proposition. First, a lemma of more general interest.

Lemma 19. *Let $(\mathcal{N} : T)$ and $(\mathcal{N} : T')$ be typings for the same $\mathcal{N}$. If T and T' are total, tight, and $\mathsf{Poly}(T) = \mathsf{Poly}(T')$, then $T = T'$.*

Proof. This follows from the construction in the proof of Proposition 16, where $\mathsf{Tight}(T)$ returns a typing which is both total and tight (and equivalent to T). □

Proposition 20 (Principal Typings Are Equivalent)**.** *If $(\mathcal{N} : T)$ and $(\mathcal{N} : U)$ are two principal typings for the same network $\mathcal{N}$, then $T \equiv U$. Moreover, if T and U are tight and total, then $T = U$.*

Proof. Both $(\mathcal{N} : T)$ and $(\mathcal{N} : U)$ are valid. Hence, by Proposition 18, both $T <: U$ and $U <: T$. This implies that $T \equiv U$. When T and U are uniformly tight, then the equality $T = U$ follows from Lemma 19. □

The next example illustrates several notions introduced earlier in this section, as well as points to issues we examine in later sections.

Example 21. The network $\mathcal{N}_1$ in Example 6 is simple enough that we can directly determine a tight, total, and principal typing T_1 for it. By easy inspection, in addition to the type assignments:

$$T_1(\varnothing) \;=\; T_1(\{a_1, a_2, a_3\}) \;=\; [0, 0],$$

[8] That the typing on the left of "<:" is less restrictive than the typing on the right of "<:" is perhaps counter-intuitive. However, from a different perspective, the typing on the left of "<:" can be viewed as the "true" typing for a flow network $\mathcal{N}$, if it is principal for $\mathcal{N}$, while the typing on the right of "<:" is only an "approximate" typing for $\mathcal{N}$. We try to adopt notation and terminology that are consistent with those in object-oriented programming.

T_1 makes six further assignments, shown below, one for every $\varnothing \subsetneq A \subsetneq \{a_1, a_2, a_3\}$. Note our conventions for writing these assignments, which we follow in later examples: input variables/arc names are listed positively, output variables/arc names are listed negatively. When $r = s = t = 0$, a principal typing T_1 for $\mathcal{N}_1$ makes the following type assignments:

$$a_1 : [0,15] \qquad a_2 : [0,35] \qquad -a_3 : [-35,0]$$
$$a_1 + a_2 : [0,35] \qquad a_1 - a_3 : [-35,0] \qquad a_2 - a_3 : [-15,0]$$

When $r = t = 0$ and $s = 10$, a principal typing T_1 for $\mathcal{N}_1$ makes the type assignments:

$$a_1 : [0,15] \qquad a_2 : [0,35] \qquad \underline{-a_3 : [-35,-10]}$$
$$\underline{a_1 + a_2 : [10,35]} \qquad a_1 - a_3 : [-35,0] \qquad a_2 - a_3 : [-15,0]$$

When $s = t = 0$ and $r = 5$, a principal typing T_1 for $\mathcal{N}_1$ makes the type assignments:

$$a_1 : [0,15] \qquad \underline{a_2 : [5,35]} \qquad \underline{-a_3 : [-35,-5]}$$
$$\underline{a_1 + a_2 : [5,35]} \qquad \underline{a_1 - a_3 : [-35,-5]} \qquad a_2 - a_3 : [-15,0]$$

The underlined assignments are those affected by the change of s from 0 to 10, or the change of r from 0 to 5. Figure 5 shows $\mathsf{Poly}(T_1)$ in all three cases.

There is considerable redundancy in T_1 in all three cases, in that several of the type assignments can be omitted without changing $\mathsf{Poly}(T_1)$. For example, when $r = s = t = 0$, a partial typing T_1' that makes only three assignments, instead of 8 by T_1, is the following:

$$a_1 : [0,15] \qquad -a_3 : [-35,0] \qquad a_1 + a_2 - a_3 : [0,0]$$

which is equivalent to T_1, *i.e.*, $\mathsf{Poly}(T_1) = \mathsf{Poly}(T_1')$. To see this, consider the diagram on the left in Figure 5: The light-shaded area on the left is the same bounded convex surface defined by both T_1 and T_1'.

And there are other partial typings besides T_1' which are equivalent to T_1 and make only three assignments. For example, T_1'' given by:

$$a_1 : [0,15] \qquad a_1 + a_2 : [0,35] \qquad a_1 + a_2 - a_3 : [0,0]$$

is also equivalent to T_1.

Figure 5 also shows the line $a_1 = a_3$ in the two-dimensional (a_1, a_3)-plane. This is the intersection of a vertical plane, call it P (not shown), containing the a_2 axis with the horizontal (a_1, a_3)-plane. P geometrically defines the requirement that the amount carried by arc a_1 is equal to that carried by a_3. This is a requirement we impose if we want to re-direct the flow out of arc a_3 and back into arc a_1. Figure 5 shows that this requirement can be satisfied when $r = s = t = 0$, or when $r = t = 0$ and $s = 10$, but not when $s = t = 0$ and $r = 5$; in the first two cases, P intersects $\mathsf{Poly}(T_1)$, but in the third, P does not intersect $\mathsf{Poly}(T_1)$. □

5 Inferring Typings that Are Total, Tight, and Principal

Let $\mathcal{N} = (\mathbf{N}, \mathbf{A})$ be a network. We follow the notation and conventions of Section 4 throughout.

Definition 22 (*How To Compute Principal Typings*)**.** Let $\mathscr{E}$ be the collection of all equations enforcing flow conservation, and $\mathscr{C}$ the collection of all inequalities enforcing capacity constraints, in $\mathcal{N}$.

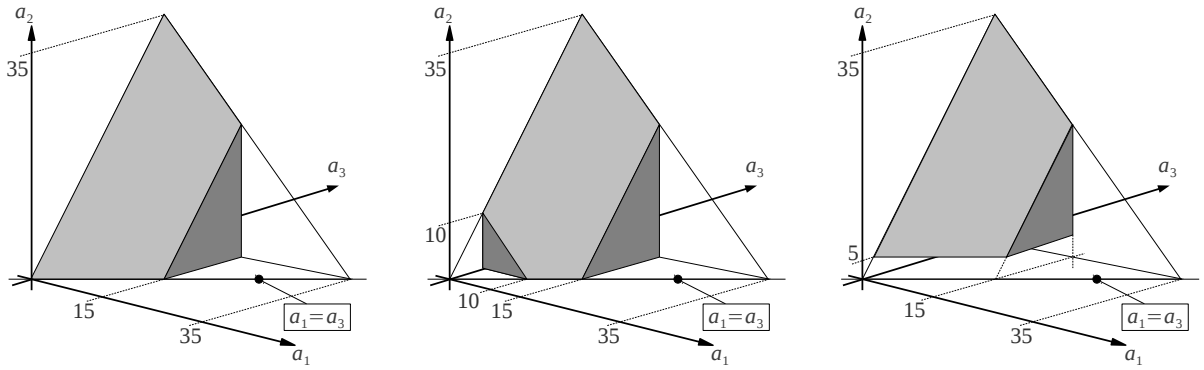

Figure 5: From Example 21, Poly(T_1), is shown as a light-shaded surface – on the left when $r = s = t = 0$, in the middle when $r = t = 0$ and $s = 10$, and on the right when $s = t = 0$ and $r = 5$.

We define the total typing $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ according to the following procedure, called **PT** (for **P**rincipal **T**yping). For every $A \in \mathscr{P}(\mathbf{A}_{\text{in,out}})$, relative to the equations and inequalities in $\mathscr{E} \cup \mathscr{C}$, we use linear programming to minimimize and maximize the same objective function:

$$\theta_A \;:=\; \sum\{\, a \mid a \in A \cap \mathbf{A}_{\text{in}} \,\} - \sum\{\, a \mid a \in A \cap \mathbf{A}_{\text{out}} \,\}$$

Relative to $\mathscr{E} \cup \mathscr{C}$, the determination of the type/interval assigned to $T(A)$ is in three steps:

1. Compute the minimum possible value $r_1 \in \mathbb{R}$ for the objective θ_A.
2. Compute the maximum possible value $r_2 \in \mathbb{R}$ for the objective θ_A.
3. Assign to $T(A)$ the interval $[r_1, r_2]$.

We write $\mathbf{PT}(\mathcal{N})$ or $\mathbf{PT}(\mathscr{E} \cup \mathscr{C})$ for the total typing T returned according to the preceding procedure. Theorem 26 confirms that $\mathbf{PT}(\mathscr{E} \cup \mathscr{C})$ is also a tight and principal typing for $\mathcal{N}$. □

Example 23. We compute a total typing T_2 for the network $\mathcal{N}_2$ shown in Figure 2 according to procedure **PT** in Definition 22. We can *either* use linear programming to compute every interval/type $T_2(A)$ – as proposed in Definition 22 – *or*, because $\mathcal{N}_2$ is fairly small, compute $T_2(A)$ by brute-force inspection, though very tediously. Following the conventions in Section 4, we use arc names as variables. Hence, for every $A \in \mathscr{P}(\mathbf{A}_{\text{in,out}})$, we write:

$$\textstyle\sum(A \cap \mathbf{A}_{\text{in}}) - \sum(A \cap \mathbf{A}_{\text{out}}) : [r_1, r_2]$$

instead of $T_2(A) = [r_1, r_2]$. The assignment of types by T_2 are $T_2(\varnothing) = [0, 0]$ and for every $A \neq \varnothing$:

$a_1 : [2, 10]$	$a_2 : [0, 7]$	$a_3 : [4, 9]$	$-a_4 : [-8, -3]$
$-a_5 : [-11, -4]$	$a_1 + a_2 : [2, 10]$	$a_1 + a_3 : [6, 14]$	$a_1 - a_4 : [-6, 7]$
$a_1 - a_5 : [-8, 4]$	$a_2 + a_3 : [4, 11]$	$a_2 - a_4 : [-8, 4]$	$a_2 - a_5 : [-11, 2]$
$a_3 - a_4 : [-4, 6]$	$a_3 - a_5 : [-7, 5]$	$-a_4 - a_5 : [-14, -7]$	$a_1 + a_2 + a_3 : [7, 14]$
$a_1 + a_2 - a_4 : [-5, 7]$	$a_1 + a_2 - a_5 : [-6, 4]$	$a_1 + a_3 - a_4 : [-2, 11]$	$a_1 + a_3 - a_5 : [-4, 8]$
$a_1 - a_4 - a_5 : [-11, -4]$	$a_2 + a_3 - a_4 : [-4, 8]$	$a_2 + a_3 - a_5 : [-7, 6]$	$a_2 - a_4 - a_5 : [-14, -6]$
$a_3 - a_4 - a_5 : [-10, -2]$	$a_1 + a_2 + a_3 - a_4 : [4, 11]$	$a_1 + a_2 + a_3 - a_5 : [3, 8]$	$a_1 + a_2 - a_4 - a_5 : [-9, -4]$
$a_1 + a_3 - a_4 - a_5 : [-7, 0]$	$a_2 + a_3 - a_4 - a_5 : [-10, -2]$	$a_1 + a_2 + a_3 - a_4 - a_5 : [0, 0]$	

As expected, the types assigned to $\mathbf{A}_{\text{in}} = \{a_1, a_2, a_3\}$ and $\mathbf{A}_{\text{out}} = \{a_4, a_5\}$ are the negations of each other, namely $[7, 14]$ and $[-14, -7]$, and demarcate the interval of feasible flows from a minimum of 7 to a maximum of 14. According to Theorem 26, the total typing T_2 as just defined is tight and principal for $\mathcal{N}_2$. □

Example 24. We use procedure **PT** in Definition 22 to infer a tight, total, and principal typing T_3 for the network $\mathcal{N}_3$ in Example 8. In addition to $T_3(\varnothing) = [0, 0]$, T_3 makes the following assignments:

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

The boxed type assignments and the underlined type assignments are for purposes of comparison with the typing T_4 in Example 25. □

Example 25. We use procedure **PT** in Definition 22 to infer a tight, total, and principal typing T_4 for the network $\mathcal{N}_4$ in Example 9. In addition to $T_4(\varnothing) = [0, 0]$, T_4 makes the following assignments:

$\boxed{a_1 : [0, 15]}$	$\boxed{a_2 : [0, 25]}$	$\boxed{-a_3 : [-15, 0]}$	$\boxed{-a_4 : [-25, 0]}$
$\boxed{a_1 + a_2 : [0, 30]}$	$\underline{a_1 - a_3 : [-10, 12]}$	$\underline{a_1 - a_4 : [-23, 15]}$	
$\underline{a_2 - a_3 : [-15, 23]}$	$\underline{a_2 - a_4 : [-12, 10]}$	$\boxed{-a_3 - a_4 : [-30, 0]}$	
$a_1 + a_2 - a_3 : [0, 25]$	$a_1 + a_2 - a_4 : [0, 15]$	$a_1 - a_3 - a_4 : [-25, 0]$	$a_2 - a_3 - a_4 : [-15, 0]$
$a_1 + a_2 - a_3 - a_4 : [0, 0]$			

In this example and the preceding one, the type assignments in rectangular boxes are for subsets of the input arcs $\{a_1, a_2\}$ and for subsets of the output arcs $\{a_3, a_4\}$, but not for subsets mixing input arcs and output arcs. Note that these are the same for the typing T_3 in Example 24 and the typing T_4 in this Example 25.

The underlined type assignments are among those that mix input and output arcs. We underline those in Example 24 that are different from the corresponding ones in this Example 25. This difference implies there are IO functions $f : \{a_1, a_2, a_3, a_4\} \to \mathbb{R}$ which can be extended to feasible flows in $\mathcal{N}_3$ (resp. in $\mathcal{N}_4$) but not in $\mathcal{N}_4$ (resp. in $\mathcal{N}_3$). This is perhaps counter-intuitive, since T_3 and T_4 make exactly the same type assignments to input arcs and, separately, output arcs (the boxed assignments). For example, the IO function f defined by:

$$f(a_1) = 15 \qquad f(a_2) = 0 \qquad f(a_3) = 3 \qquad f(a_4) = 12$$

can be extended to a feasible flow in $\mathcal{N}_4$ but not in $\mathcal{N}_3$. The reason is that $f(a_1) - f(a_3) = 12$ violates (*i.e.*, is outside) the type $T_3(\{a_1, a_3\}) = [-10, 10]$. Similarly, the IO function f defined by:

$$f(a_0) = 0 \qquad f(a_2) = 25 \qquad f(a_3) = 0 \qquad f(a_4) = 25$$

can be extended to a feasible flow in $\mathcal{N}_3$ but not in $\mathcal{N}_4$, the reason being that $f(a_2) - f(a_3) = 25$ violates the type $T_4(\{a_2, a_3\}) = [-15, 23]$. From the preceding, neither T_3 nor T_4 is a subtyping of the other, in the sense explained in Subsection 4.2. □

The proof of Theorem 26 is delayed to Appendix A. Our proof here is based on a pre-processing of the given network $\mathcal{N} = (\mathbf{N}, \mathbf{A})$, which transforms it into an equivalent so-called "augmented network" $\mathcal{N}' = (\mathbf{N}', \mathbf{A}')$,

where capacity constraints are more general than those in $\mathcal{N}$ and self-loops are allowed. We build $\mathcal{N}'$ from $\mathcal{N}$ so that $\mathbf{A}'_{\text{in,out}} = \mathbf{A}_{\text{in,out}}$ and $\mathbf{A}'_{\#} \subseteq \mathbf{A}_{\#}$, and in such a way that all the remaining internal arcs in $\mathbf{A}'_{\#}$ are self-loops, thus essentially making the set of feasible flows in $\mathcal{N}'$ and the set of feasible IO functions in $\mathcal{N}'$ coincide.[9]

Theorem 26 (Inferring Total, Tight, and Principal Typings)**.** *The typing* $T : \mathscr{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$ *returned by* $\mathbf{PT}(\mathscr{E} \cup \mathscr{C})$ *in Definition 22 is total, tight and principal for network* $\mathcal{N}$*.*

References

[AMM01] H. Aydin, R. Melhem, and D. Moss. Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics. In *Proc. of EuroMicro Conference on Real-Time Systems*, pages 225–232, 2001.

[AMO93] R.K. Ahuja, T. L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, N.J., 1993.

[BK11] A. Bestavros and A. Kfoury. A Domain-Specific Language for Incremental and Modular Design of Large-Scale Verifiably-Safe Flow Networks. In *Proc. of IFIP Working Conference on Domain-Specific Languages (DSL 2011), EPTCS Volume 66*, pages 24–47, Sept 2011.

[BKLO09] A. Bestavros, A. Kfoury, A. Lapets, and M. Ocean. Safe Compositional Network Sketches: Tool and Use Cases. In *IEEE Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Wash D.C., December 2009.

[BKLO10] A. Bestavros, A. Kfoury, A. Lapets, and M. Ocean. Safe Compositional Network Sketches: The Formal Framework. In *13th ACM HSCC*, Stockholm, April 2010.

[BV09] S. Boyd and L. Vanderberghe. *Convex Optimization*. Cambridge University Press, New York, USA, (second printing with corrections) 2009.

[Cun76] W.H. Cunningham. A Network Simplex Method. *Mathematical Programming*, 11:105–116, 1976.

[Cun79] W. H. Cunningham. Theoretical Properties of the Network Simplex Method. *Mathematics of Operations Research*, 4(2):196–208, May 1979.

[DL97] Z. Deng and J. W.-S. Liu. Scheduling Real-Time Applications in an Open Environment. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, pages 308–319, 1997.

[Jim95] Trevor Jim. What Are Principal Typings and What Are They Good For? Tech. memo. MIT/LCS/TM-532, MIT, 1995.

[Jim96] Trevor Jim. What Are Principal Typings and What Are They Good For? In *Proc. of 23rd ACM Symp. on Principles of Programming Languages*, pages 42–53, 1996.

[KBS04] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[Kfo11a] A. Kfoury. A Domain-Specific Language for Incremental and Modular Design of Large-Scale Verifiably-Safe Flow Networks (Part 1). Technical Report BUCS-TR-2011-011, CS Dept, Boston Univ, May 2011.

[9] Although based on an initial graph-transformation, the proof in this report is heavily algebraic. Alternative, less algebra-oriented proofs of Theorem 26 are also possible. In [Kfo12], we propose an approach based on decomposing the given network $\mathcal{N}$ into a set of full acyclic paths from input arcs to output arcs, similar to the examination in Subsection **??** below.

[Kfo11b] A. Kfoury. The Denotational, Operational, and Static Semantics of a Domain-Specific Language for the Design of Flow Networks. In *Proc. of SBLP 2011: Brazilian Symposium on Programming Languages*, Sept 2011.

[Kfo12] A. Kfoury. Algebraic Characterizations of Flow-Network Typings. Technical Report BUCS-TR-2012-004, CS Dept, Boston Univ, February 2012.

[KM12] A. Kfoury and Saber Mirzaei. A Different Approach to the Design and Analysis of Network Algorithms. Technical Report BUCS-TR-2012-019, CS Dept, Boston Univ, December 2012.

[LBJ+95] S.S. Lim, Y.H. Bae, G.T. Jang, B.D. Rhee, S.L. Min, C.Y. Park, H.S., K. Park, S.M. Moon, and C.S. Kim. An Accurate Worst Case Timing Analysis for RISC Processors. In *IEEE REAL-TIME SYSTEMS SYMPOSIUM*, pages 97–108, 1995.

[PLS01] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic Voltage Scaling on a Low-Power Microprocessor. In *Mobile Computing and Networking - Mobicom*, pages 251–259, 2001.

[Reg02] J. Regehr. Inferring Scheduling Behavior with Hourglass. In *Proc. of the USENIX Annual Technical Conf. FREENIX Track*, pages 143–156, 2002.

[SBKL11] N. Soule, A. Bestavros, A. Kfoury, and A. Lapets. Safe Compositional Equation-based Modeling of Constrained Flow Networks. In *Proc. of 4th Int'l Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Zürich, September 2011.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization, New York, USA, 1986.

[Sch12] A. Schrijver. *A Course in Combinatorial Optimization*. CWI (the manuscript can be downloaded from the author's webpage `http://homepages.cwi.nl/~lex/`), Amsterdam, The Netherlands, 2012.

[SL03] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *24th IEEE International Real-Time Systems Symposium (RTSS '03)*, page 2, Washington, DC, USA, 2003. IEEE Comp Soc.

[SLM98] D.C. Schmidt, D.L. Levine, and S. Mungee. The Design of the tao real-time object request broker. *Computer Communications*, 21:294–324, 1998.

[Sta00] J.A. Stankovic. VEST: A Toolset for Constructing and Analyzing Component Based Embedded Systems. In *Proc. EMSOFT01, LNCS 2211*, pages 390–402. Springer, 2000.

A Remaining Proofs for Section 5

We use the notation and assumptions of Section 5. By the comments in Subsection 2.2, with no loss of generality, we can assume there are no consumer nodes and no producer nodes in networks, and that all nodes are transshipment nodes.

Definition 27 (*Augmented Networks*)**.** Let $\mathcal{N} = (\mathbf{N}, \mathbf{A})$ be a network, with $\mathscr{E}$ the set of flow-conservation equations (one for every node $\nu \in \mathbf{N}$) and $\mathscr{C}$ the set of capacity constraints (one for every arc $a \in \mathbf{A}$, in the form of two inequalities specifying an interval of permissible values for a), as in Definitions 1 and 2.

An *auxiliary capacity constraint* is a generalization of a capacity constraint in that it specifies an interval of permissible values, possibly negative, for a summation of arcs/variables from $\mathbf{A}$:

$$r \leqslant \sum A - \sum B \leqslant s$$

where $r, s \in \mathbb{R}$ with $r \leqslant s$, and $A, B \subseteq \mathbf{A}$ with $A \cap B = \varnothing$ and $|A \cup B| \geqslant 1$.

$\mathcal{N}$ is an *augmented network* if $\mathcal{N}$ is supplied with a finite set $\mathscr{C}^*$ of auxiliary capacity constraints. Moreover, in contrast to the definition of (non-augmented) networks in Section 2, an augmented network may contain self-loops, *i.e.*, arcs a such that $\mathit{tail}(a) = \mathit{head}(a)$, as well as multiple arcs between the same two nodes, *i.e.*, arcs a and a' such that $a \neq a'$, $\mathit{tail}(a) = \mathit{tail}(a')$ and $\mathit{head}(a) = \mathit{head}(a')$.

A flow $f : \mathbf{A} \to \mathbb{R}_+$ in the augmented network $\mathcal{N}$ is a *feasible flow* if f satisfies the flow-conservation equations $\mathscr{E}$, the capacity constraints $\mathscr{C}$, and the auxiliary capacity constraints $\mathscr{C}^*$.

To make explicit the flow-convervation equations, capacity constraints, and auxiliary capacity constraitns, of an augmented network $\mathcal{N}$, we may write $\mathcal{N} = (\mathbf{N}, \mathbf{A}, \mathscr{E}, \mathscr{C}, \mathscr{C}^*)$ instead of $\mathcal{N} = (\mathbf{N}, \mathbf{A})$. □

Let $\mathcal{N} = (\mathbf{N}, \mathbf{A}, \mathscr{E}, \mathscr{C})$ be a fixed (non-augmented) network, with $\mathbf{A} = \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} \uplus \mathbf{A}_{\#}$ where $|\mathbf{A}_{\#}| = p \geqslant 0$. Starting from $\mathcal{N}$, we construct a sequence of $p + 1$ augmented networks in $p + 1$ stages:

$$\mathcal{N}_1 = (\mathbf{N}_1, \mathbf{A}_1, \mathscr{E}_1, \mathscr{C}_1, \mathscr{C}_1^*),\ \mathcal{N}_2 = (\mathbf{N}_2, \mathbf{A}_2, \mathscr{E}_2, \mathscr{C}_2, \mathscr{C}_2^*),\ \ldots\ ,\ \mathcal{N}_{p+1} = (\mathbf{N}_{p+1}, \mathbf{A}_{p+1}, \mathscr{E}_{p+1}, \mathscr{C}_{p+1}, \mathscr{C}_{p+1}^*).$$

For every $1 \leqslant k \leqslant p+1$, the set of arcs $\mathbf{A}_k$ is partitioned as $\mathbf{A}_k = \mathbf{A}_{k,\text{in}} \uplus \mathbf{A}_{k,\text{out}} \uplus \mathbf{A}_{k,\#}$ – input arcs, output arcs, and internal arcs – with $\mathbf{A}_{k,\text{in,out}} = \mathbf{A}_{k,\text{in}} \cup \mathbf{A}_{k,\text{out}}$. These augmented networks are related as follows:

1. $\mathbf{N}_1 = \mathbf{N}$, $\mathbf{A}_1 = \mathbf{A}$, $\mathscr{E}_1 = \mathscr{E}$, $\mathscr{C}_1 = \mathscr{C}$, and $\mathscr{C}_1^* = \varnothing$.
2. $\mathbf{N}_1 \supseteq \mathbf{N}_2 \supseteq \cdots \supseteq \mathbf{N}_{p+1}$.
3. $\mathbf{A}_{1,\text{in,out}} = \mathbf{A}_{2,\text{in,out}} = \cdots = \mathbf{A}_{p+1,\text{in,out}}$ and $\mathbf{A}_{1,\#} \supseteq \mathbf{A}_{2,\#} \supseteq \cdots \supseteq \mathbf{A}_{p+1,\#}$.

Stage 1 defines the first augmented network $\mathcal{N}_1$ according to point 1 above. At stage $k + 1$, with $1 \leqslant k \leqslant p$, the construction of $\mathcal{N}_{k+1}$ from $\mathcal{N}_k$ consists in processing an internal arc in $\mathbf{A}_{k,\#}$, which may or may not be deleted as a result. The internal arc which is processed at stage $k + 1$ is selected arbitrarily and can be any member of $\mathbf{A}_{k,\#}$. Let:

$$\mathbf{A}_{1,\#} = \mathbf{A}_{\#} = \{b_1, b_2, \ldots, b_p\},$$

where the b_k's are indexed, from left to right, in the order in which they are processed. When we process arc b_k, if $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$, we merge node $\mathit{head}(b_k)$ into node $\mathit{tail}(b_k)$, thus deleting node $\mathit{head}(b_k)$ but keeping $\mathit{tail}(b_k)$. We therefore have:

4. For every $1 \leqslant k \leqslant p$:

$$\mathbf{N}_{k+1} := \begin{cases} \mathbf{N}_k & \text{if } \mathit{tail}(b_k) = \mathit{head}(b_k), \\ \mathbf{N}_k - \{\mathit{head}(b_k)\} & \text{if } \mathit{tail}(b_k) \neq \mathit{head}(b_k). \end{cases}$$

5. For every $1 \leqslant k \leqslant p$:

$$\mathbf{A}_{k+1,\#} \;:=\; \begin{cases} \mathbf{A}_{k,\#} & \text{if } \mathit{tail}(b_k) = \mathit{head}(b_k), \\ \mathbf{A}_{k,\#} - \{b_k\} & \text{if } \mathit{tail}(b_k) \neq \mathit{head}(b_k). \end{cases}$$

6. For every $1 \leqslant k \leqslant p$:

$$\mathscr{C}_{k+1} \;:=\; \begin{cases} \mathscr{C}_k & \text{if } \mathit{tail}(b_k) = \mathit{head}(b_k), \\ \mathscr{C}_k - \{\, \underline{\mathbf{c}}(b_k) \leqslant b_k \leqslant \overline{\mathbf{c}}(b_k) \,\} & \text{if } \mathit{tail}(b_k) \neq \mathit{head}(b_k). \end{cases}$$

At the end of stage k, with $1 \leqslant k \leqslant p$, let $\mathit{tail}(b_k) = \nu_1$ and $\mathit{head}(b_k) = \nu_2$. The flow-conservation equations at nodes ν_1 and ν_2 are E_{ν_1} and E_{ν_2}, respectively:

$$(E_{\nu_1}) \qquad \sum\{\, a \in \mathbf{A}_k \mid \mathit{head}(a) = \nu_1 \,\} \;=\; \sum\{\, a \in \mathbf{A}_k \mid \mathit{tail}(a) = \nu_1 \,\},$$

$$(E_{\nu_2}) \qquad \sum\{\, a \in \mathbf{A}_k \mid \mathit{head}(a) = \nu_2 \,\} \;=\; \sum\{\, a \in \mathbf{A}_k \mid \mathit{tail}(a) = \nu_2 \,\}.$$

If $\nu_1 = \nu_2 = \nu$, then b_k is a self-loop at node ν and appears on both sides of equation $E_\nu = E_{\nu_1} = E_{\nu_2}$. Let E'_ν be the simplification of E_ν obtained by crossing out the left occurrence and the right occurrence of b_k in E_ν.

7.1. For every $1 \leqslant k \leqslant p$, if $\mathit{tail}(b_k) = \mathit{head}(b_k) = \nu$, then:

$$\mathscr{E}_{k+1} \;:=\; (\mathscr{E}_k - \{E_\nu\}) \cup \{E'_\nu\}.$$

7.2. For every $1 \leqslant k \leqslant p$, if $\mathit{tail}(b_k) = \nu_1 \neq \nu_2 = \mathit{head}(b_k)$, then:

$$\mathscr{E}_{k+1} := \big(\mathscr{E}_k - \{\, E_{\nu_1}, E_{\nu_2} \,\}\big) \bigcup \Big\{\, \sum\{\, a \in \mathbf{A}_k - \{b_k\} \mid \mathit{head}(a) \in \{\nu_1, \nu_2\} \,\} = \sum\{\, a \in \mathbf{A}_k - \{b_k\} \mid \mathit{tail}(a) \in \{\nu_1, \nu_2\} \,\} \,\Big\}.$$

We update the functions $\mathit{tail}()$ and $\mathit{head}()$ as follows. For every $a \in \mathbf{A}_k - \{b_k\}$:

$$\mathit{tail}(a) \;:=\; \begin{cases} \nu_1 & \text{if } \mathit{tail}(a) = \nu_2, \\ \mathit{tail}(a) & \text{otherwise,} \end{cases}$$

$$\mathit{head}(a) \;:=\; \begin{cases} \nu_1 & \text{if } \mathit{head}(a) = \nu_2, \\ \mathit{head}(a) & \text{otherwise.} \end{cases}$$

When $\mathit{tail}(b_k) = \nu_1 \neq \nu_2 = \mathit{head}(b_k)$, the definition of $\mathscr{E}_{k+1}$ excludes the two flow-conservation equations in $\mathscr{E}_k$ that mention b_k and includes instead a new flow-conservation equation that omits b_k. This new equation may mention the same arc/variable b_ℓ on both sides of "=", for some $k < \ell \leqslant p$, which happens when $\mathit{tail}(b_\ell) = \nu_1$ and $\mathit{head}(b_\ell) = \nu_2$ or when $\mathit{tail}(b_\ell) = \nu_2$ and $\mathit{head}(b_\ell) = \nu_1$, *i.e.*, b_ℓ is now a self-loop at node ν_1 which will be eliminated when going from $\mathscr{E}_\ell$ to $\mathscr{E}_{\ell+1}$.

It remains to define $\mathscr{C}^*_{k+1}$ from $\mathscr{C}^*_k$ at stage $k+1$.

8.1. For every $1 \leqslant k \leqslant p$, if $\mathit{tail}(b_k) = \mathit{head}(b_k)$, then:

$$\mathscr{C}^*_{k+1} \;:=\; \mathscr{C}^*_k.$$

If $\mathit{tail}(b_k) = \nu_1 \neq \nu_2 = \mathit{head}(b_k)$, we need to consider the flow-conservation equations that mention b_k more carefully. We can write the flow-conservation equations E_{ν_1} and E_{ν_2} as follows:

$$(E_{\nu_1}) \qquad \sum\{\, a \in \mathbf{A}_k \mid \mathit{head}(a) = \nu_1 \,\} \;=\; b_k + \sum\{\, a \in \mathbf{A}_k \mid \mathit{tail}(a) = \nu_1 \text{ and } a \neq b_k \,\},$$

$$(E_{\nu_2}) \qquad b_k + \sum\{\, a \in \mathbf{A}_k \mid \mathit{head}(a) = \nu_2 \text{ and } a \neq b_k \,\} \;=\; \sum\{\, a \in \mathbf{A}_k \mid \mathit{tail}(a) = \nu_2 \,\},$$

from which we get the two following equations:

$$\begin{aligned} +b_k \;&=\; \varphi_1 \quad \text{where} \\ \varphi_1 \;&:=\; +\sum\{\, a \in \mathbf{A}_k \mid \mathit{head}(a) = \nu_1 \,\} - \sum\{\, a \in \mathbf{A}_k \mid \mathit{tail}(a) = \nu_1 \text{ and } a \neq b_k \,\}, \\ -b_k \;&=\; \varphi_2 \quad \text{where} \\ \varphi_2 \;&:=\; +\sum\{\, a \in \mathbf{A}_k \mid \mathit{head}(a) = \nu_2 \text{ and } a \neq b_k \,\} - \sum\{\, a \in \mathbf{A}_k \mid \mathit{tail}(a) = \nu_2 \,\}. \end{aligned}$$

It is important to note that, in both φ_1 and φ_2, arcs *entering* a node (ν_1 or ν_2) are included positively and arcs *exiting* a node (ν_1 or ν_2) are included negatively. For the case $\nu_1 \neq \nu_2$, we define the substitution σ_k as follows:

$$\sigma_k \;:=\; \big\{\, +b_k \mapsto \varphi_1,\; -b_k \mapsto \varphi_2 \,\big\}.$$

Observe we make a distinction between occurrences of b_k appearing positively and those appearing negatively: "$+b_k$" and "$-b_k$" are mapped to two distinct expressions, φ_1 and φ_2, that guarantee that entering (resp., exiting) arcs/variables appear positively (resp., negatively).

8.2. For every $1 \leqslant k \leqslant p$, if $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$, then:

$$\mathscr{C}^*_{k+1} \;:=\; \sigma_k(\mathscr{C}^*_k) \bigcup \big\{\, \underline{\boldsymbol{c}}(b_k) \leqslant \varphi_1 \leqslant \overline{\boldsymbol{c}}(b_k),\; -\overline{\boldsymbol{c}}(b_k) \leqslant \varphi_2 \leqslant -\underline{\boldsymbol{c}}(b_k) \,\big\}.$$

The substitution part $\sigma_k(\,\mathscr{C}^*_k\,)$ in $\mathscr{C}^*_{k+1}$ eliminates all occurrences of "$+b_k$" and "$-b_k$", while the remaining part in $\mathscr{C}^*_{k+1}$ includes two new auxiliary constraints that do not mention "$+b_k$" and "$-b_k$".

In the proofs to follow, we say "the construction of augmented networks" or "the sequence of augmented networks induced by $\mathcal{N}$" to refer to the sequence $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{p+1}$ and the way the entries in it are defined.

Lemma 28. *For every $1 \leqslant k \leqslant p$:*

1. *If $f : \mathbf{A}_k \to \mathbb{R}_+$ is a feasible flow in $\mathcal{N}_k$, then $g = \big[\!\big[f \big]\!\big]_{\mathbf{A}_{k+1}} : \mathbf{A}_{k+1} \to \mathbb{R}_+$ is a feasible flow in $\mathcal{N}_{k+1}$.*
2. *If $g : \mathbf{A}_{k+1} \to \mathbb{R}_+$ is a feasible flow in $\mathcal{N}_{k+1}$, then there is a feasible flow $f : \mathbf{A}_k \to \mathbb{R}_+$ in $\mathcal{N}_k$ such that $g = \big[\!\big[f \big]\!\big]_{\mathbf{A}_{k+1}}$.*

Proof. Throughout this proof, let $\varphi := \varphi_1$, where φ_1 is defined in point 8.2 in the construction of augmented networks. (Alternatively, we can also take $\varphi := -\varphi_2$; the proof works again.)

For part 1, suppose $f : \mathbf{A}_k \to \mathbb{R}_+$ is feasible in $\mathcal{N}_k$. This means f satisfies $\mathscr{E}_k$, $\mathscr{C}_k$, and $\mathscr{C}^*_k$. Let $g = \big[\!\big[f \big]\!\big]_{\mathbf{A}_{k+1}}$. Consider the two cases $\mathit{tail}(b_k) = \mathit{head}(b_k)$ and $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$ separately.

If $\mathit{tail}(b_k) = \mathit{head}(b_k)$, then it is immediate that g satisfies $\mathscr{E}_{k+1}$, $\mathscr{C}_{k+1}$, and $\mathscr{C}^*_{k+1}$, because $\mathbf{A}_k = \mathbf{A}_{k+1}$ and $g = \big[\!\big[f \big]\!\big]_{\mathbf{A}_{k+1}} = f$. If $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$, then it is easy to see that g satisfies $\mathscr{E}_{k+1}$ and $\mathscr{C}_{k+1}$. By points 7.2 and 8.2 in the construction of augmented networks, g also satisfies $\mathscr{C}^*_{k+1}$, because φ does not mention b_k and f satisfies $\mathscr{E}_k$.

For part 2, suppose $g : \mathbf{A}_{k+1} \to \mathbb{R}_+$ is feasible in $\mathcal{N}_{k+1}$. This means g satisfies $\mathscr{E}_{k+1}$, $\mathscr{C}_{k+1}$, and $\mathscr{C}^*_{k+1}$. Again, consider the two cases $\mathit{tail}(b_k) = \mathit{head}(b_k)$ and $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$ separately. For the case when $\mathit{tail}(b_k) = \mathit{head}(b_k) = \nu$, the desired conclusion is immediate, because $\mathbf{A}_k = \mathbf{A}_{k+1}$ and we define $f := g$.

For the case when $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$, we use the notation and definitions of points 7.2 and 8.2 in the construction of augmented networks. In this case, $g : (\mathbf{A}_k - \{b_k\}) \to \mathbb{R}_+$. Because g satisfies $\mathscr{E}_{k+1}$ in point 7.2,

it is readily checked that $g(\varphi) = g(\varphi_1) = -g(\varphi_2)$, where "$g(\varphi)$" denotes the value obtained by applying g to every arc/variable occurring in the expression φ. We define the extension $f : \mathbf{A}_k \to \mathbb{R}_+$ by letting $f(b_k) = g(\varphi)$. It is now straightforward to check that, given that g satisfies $\mathscr{E}_{k+1}$, $\mathscr{C}_{k+1}$, and $\mathscr{C}^*_{k+1}$, so does f satisfies $\mathscr{E}_k$, $\mathscr{C}_k$, and $\mathscr{C}^*_k$, by point 7.2, point 6, and point 8.2 in the construction of augmented networks, respectively. □

Definition 29 (*Extreme Flows*)**.** Let $f_1, f_2 : \mathbf{A} \to \mathbb{R}_+$ be flows in augmented network $\mathcal{N} = (\mathbf{N}, \mathbf{A}, \mathscr{E}, \mathscr{C}, \mathscr{C}^*)$, where $\mathbf{A} = \{a_1, \ldots, a_q\}$. We use $a_1, \ldots, a_q$ as the q dimensions, in this order, of the space $\mathbb{R}^q$. The *addition* of f_1 and f_2 is defined pointwise as usual, viewing them as q-dimensional vectors:

$$f_1 + f_2 \;=\; \langle f_1(a_1) + f_2(a_1),\; \ldots\; , f_1(a_q) + f_2(a_q)\rangle.$$

A flow $f : \mathbf{A} \to \mathbb{R}_+$ is an *extreme (feasible) flow* in $\mathcal{N}$ if there do not exist feasible flows $f_1, f_2 : \mathbf{A} \to \mathbb{R}_+$ in $\mathcal{N}$ such that $f_1 \neq f_2$ and $f = (f_1 + f_2)/2$.

It is useful to consider an alternative geometric formulation of extreme flows. Let $\mathscr{E}$, $\mathscr{C}$, and $\mathscr{C}^*$, be the flow-conservation equations, the capacity constraints, and the auxiliary capacity constraints of $\mathcal{N}$. The combined set $\mathscr{E} \cup \mathscr{C} \cup \mathscr{C}^*$ defines a polytope in the hyperspace $\mathbb{R}^q$, which we denote $\mathsf{Poly}(\mathscr{E} \cup \mathscr{C} \cup \mathscr{C}^*)$. The extreme flows in $\mathcal{N}$ are precisely the *extreme points* (or *vertices*) of $\mathsf{Poly}(\mathscr{E} \cup \mathscr{C} \cup \mathscr{C}^*)$. □

The following lemma refines the preceding one.

Lemma 30. *For every* $1 \leqslant k \leqslant p$*:*

1. *If* $f : \mathbf{A}_k \to \mathbb{R}_+$ *is an extreme flow in* $\mathcal{N}_k$*, then* $g = \big[f\big]_{\mathbf{A}_{k+1}} : \mathbf{A}_{k+1} \to \mathbb{R}_+$ *is an extreme flow in* $\mathcal{N}_{k+1}$*.*
2. *If* $g : \mathbf{A}_{k+1} \to \mathbb{R}_+$ *is an extreme flow in* $\mathcal{N}_{k+1}$*, then there is an extreme flow* $f : \mathbf{A}_k \to \mathbb{R}_+$ *in* $\mathcal{N}_k$ *such that* $g = \big[f\big]_{\mathbf{A}_{k+1}}$*.*

Part 1 is the more difficult, because the projection on $\mathbb{R}^{|\mathbf{A}_{k+1}|}$ of the extreme point of an arbitrary polytope P in the space $\mathbb{R}^{|\mathbf{A}_k|}$ is not generally an extreme point of the projection $\big[P\big]_{\mathbf{A}_{k+1}}$.

Proof. Consider the two cases $\mathit{tail}(b_k) = \mathit{head}(b_k)$ and $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$ separately. For the case $\mathit{tail}(b_k) = \mathit{head}(b_k)$, both parts of the lemma are immediate because $\mathbf{A}_k = \mathbf{A}_{k+1}$.

For the case $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$, let $\mathbf{A}_k = \{a_1, \ldots, a_{q+1}\}$ with $b_k = a_{q+1}$, so that $\mathbf{A}_{k+1} = \{a_1, \ldots, a_q\}$. We assume the arcs/variables are ordered according to their indices from 1 to q. Hence, if $g : \mathbf{A}_{k+1} \to \mathbb{R}_+$ is a flow in $\mathcal{N}_{k+1}$, then:

$$g \;=\; \langle\, g(a_1),\; g(a_2),\; \ldots\; ,\; g(a_q)\,\rangle \;=\; \langle\, r_1,\; r_2,\; \ldots\; ,\; r_q\,\rangle$$

is a point in the q-dimensional space $\mathbb{R}^q$, for some $r_1, r_2, \ldots, r_q \in \mathbb{R}_+$. Let $f : \mathbf{A}_k \to \mathbb{R}_+$ be the flow in $\mathcal{N}_k$ defined from g as in the proof for part 2 in Lemma 28. We can therefore write:

$$f \;=\; \langle\, r_1,\; r_2,\; \ldots\; ,\; r_q,\; g(\varphi)\,\rangle$$

where φ is as in the proof of Lemma 28. Hence, flow $g \in \mathsf{Poly}(\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1})$ is the projection of flow $f \in \mathsf{Poly}(\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k)$ on the subspace $\mathbb{R}^q$ of the space $\mathbb{R}^{q+1}$. It follows that if g is an extreme point of $\mathsf{Poly}(\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1})$, then f is an extreme point of $\mathsf{Poly}(\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k)$. This completes the proof of part 2 when $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$.

It remains to prove part 1 when $\mathit{tail}(b_k) \neq \mathit{head}(b_k)$. We prove the contrapositive: If $g = \big[f\big]_{\mathbf{A}_{k+1}}$ is not an extreme point of $\mathsf{Poly}(\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1})$, then f is not an extreme point of $\mathsf{Poly}(\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k)$. If g is not an extreme point, there are flows $g', g'' : \mathbf{A}_{k+1} \to \mathbb{R}_+$ in $\mathcal{N}_{k+1}$ such that $g' \neq g''$ and $g = (g' + g'')/2$. Let:

$$g' \;=\; \langle\, r'_1,\; r'_2,\; \ldots\; ,\; r'_q\,\rangle \quad \text{and} \quad g'' \;=\; \langle\, r''_1,\; r''_2,\; \ldots\; ,\; r''_q\,\rangle.$$

From g' and g'', define the flows $f' : \mathbf{A}_k \to \mathbb{R}_+$ and $f'' : \mathbf{A}_k \to \mathbb{R}_+$ in $\mathcal{N}_k$, respectively, as in the proof of Lemma 28. Hence:

$$f' \;=\; \langle\, r'_1,\, r'_2,\, \ldots\,,\, r'_q,\, g'(\varphi)\,\rangle \quad \text{and} \quad f'' \;=\; \langle\, r''_1,\, r''_2,\, \ldots\,,\, r''_q,\, g''(\varphi)\,\rangle.$$

We already have that:

$$r_1 = (r'_1 + r''_1)/2,\; r_2 = (r'_2 + r''_2)/2,\; \ldots\,,\; r_q = (r'_q + r''_q)/2$$

because g is not an extreme point. It therefore suffices to prove that $g(\varphi) = \big(g'(\varphi) + g''(\varphi)\big)/2$ in order to conclude that $f = (f' + f'')/2$, *i.e.*, f is not an extreme point. Let $\varphi = \sum A - \sum B$, where $A, B \subseteq \mathbf{A}_k$. We have:

$$\begin{aligned}
g(\varphi) \;&=\; \textstyle\sum \{\, g(a) \mid a \in A \,\} \;-\; \sum \{\, g(b) \mid b \in B \,\} \\
&=\; \textstyle\sum \{\, (g'(a) + g''(a))/2 \mid a \in A \,\} \;-\; \sum \{\, (g'(b) + g''(b))/2 \mid b \in B \,\} \\
&=\; \big(\textstyle\sum \{\, g'(a) \mid a \in A \,\} - \sum \{\, g'(b) \mid b \in B \,\}\big)/2 \;+\; \big(\sum \{\, g''(a) \mid a \in A \,\} - \sum \{\, g''(b) \mid b \in B \,\}\big)/2 \\
&=\; \big(g'(\varphi) \;+\; g''(\varphi)\big)/2,
\end{aligned}$$

which is the desired conclusion and completes the proof. □

We can run procedure $\mathbf{PT}$ in Definition 22 on input $\mathscr{E} \cup \mathscr{C} \cup \mathscr{C}^*$ to infer a total and tight typing T – soon to be shown principal – for the augmented network $\mathcal{N}$, in which case we may write $T = \mathbf{PT}(\mathscr{E} \cup \mathscr{C} \cup \mathscr{C}^*)$. As in Section 4, T defines a polytope in the hypersapce $\mathbb{R}^{|\mathbf{A}_{\text{in,out}}|}$, which we denote $\mathsf{Poly}(T)$.

Lemma 31. *Let $\mathcal{N} = (\mathbf{N}, \mathbf{A})$ be a non-augmented network. For the sequence of augmented networks induced by $\mathcal{N}$, namely, $\mathcal{N}_k = (\mathbf{N}_k, \mathbf{A}_k, \mathscr{E}_k, \mathscr{C}_k, \mathscr{C}^*_k)$ with $1 \leqslant k \leqslant p+1$, we have:*

1. $\mathbf{PT}(\mathscr{E}_1 \cup \mathscr{C}_1 \cup \mathscr{C}^*_1) = \mathbf{PT}(\mathscr{E}_2 \cup \mathscr{C}_2 \cup \mathscr{C}^*_2) = \;\cdots\; = \mathbf{PT}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})$,
2. $\big[\mathsf{Poly}(\mathscr{E}_1 \cup \mathscr{C}_1 \cup \mathscr{C}^*_1)\big]_{\mathbf{A}_{\text{in,out}}} = \big[\mathsf{Poly}(\mathscr{E}_2 \cup \mathscr{C}_2 \cup \mathscr{C}^*_2)\big]_{\mathbf{A}_{\text{in,out}}} = \;\cdots\; = \big[\mathsf{Poly}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})\big]_{\mathbf{A}_{\text{in,out}}}$.

Proof. For part 1, it suffices to show $\mathbf{PT}(\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k) = \mathbf{PT}(\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1})$ for arbitrary $1 \leqslant k \leqslant p$. Consider how procedure $\mathbf{PT}$ assigns an interval/type $[r_1, r_2]$ to a set $A \subseteq \mathbf{A}_{\text{in,out}}$. The end points r_1 and r_2 are the minimum and maximum, respectively, of the objective function:

$$\theta_A \;=\; \textstyle\sum \{\, a \mid a \in A \cap \mathbf{A}_{\text{in}} \,\} - \sum \{\, a \mid a \in A \cap \mathbf{A}_{\text{out}} \,\}$$

relative to $\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k$ for $\mathcal{N}_k$, or relative to $\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1}$ for $\mathcal{N}_{k+1}$. By standard considerations of linear programming, these minimum and maximum values are attained at extreme flows/extreme points of $\mathsf{Poly}(\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k)$ or $\mathsf{Poly}(\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1})$, respectively. The desired conclusion follows by Lemma 30.

For part 2, the conclusion of Lemma 28 can be written equivalently as follows, for every $1 \leqslant k \leqslant p$:

$$\mathsf{Poly}(\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1}) \;=\; \big[\mathsf{Poly}(\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k)\big]_{\mathbf{A}_{k+1}}.$$

Hence,

$$\big[\mathsf{Poly}(\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1})\big]_{\mathbf{A}_{\text{in,out}}} \;=\; \big[\big[\mathsf{Poly}(\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k)\big]_{\mathbf{A}_{k+1}}\big]_{\mathbf{A}_{\text{in,out}}},$$

which, given that $\mathbf{A}_{k+1} \supseteq \mathbf{A}_{\text{in,out}}$, in turn implies:

$$\big[\mathsf{Poly}(\mathscr{E}_{k+1} \cup \mathscr{C}_{k+1} \cup \mathscr{C}^*_{k+1})\big]_{\mathbf{A}_{\text{in,out}}} \;=\; \big[\mathsf{Poly}(\mathscr{E}_k \cup \mathscr{C}_k \cup \mathscr{C}^*_k)\big]_{\mathbf{A}_{\text{in,out}}}.$$

The desired conclusion for part 2 follows. □

Lemma 32. *Let $\mathcal{N} = (\mathbf{N}, \mathbf{A})$ be a non-augmented network and let $\mathcal{N}_{p+1} = (\mathbf{N}_{p+1}, \mathbf{A}_{p+1}, \mathscr{E}_{p+1}, \mathscr{C}_{p+1}, \mathscr{C}^*_{p+1})$ be the last augmented network in the sequence of augmented networks induced by $\mathcal{N}$. Then all the internal arcs of $\mathcal{N}_{p+1}$ are self-loops.*

Proof. Straightforward from the construction of augmented networks induced by $\mathcal{N}$. □

If $\mathcal{N}$ is connected as a directed graph, it is easy to see that there is exactly one node, say ν, in the last augmented network $\mathcal{N}_{p+1}$. This in turn implies that:

- If $a \in \mathbf{A}_{p+1,\text{in}}$, then $\mathit{head}(a) = \nu$,
- If $a \in \mathbf{A}_{p+1,\text{out}}$, then $\mathit{tail}(a) = \nu$,
- If $a \in \mathbf{A}_{p+1,\#}$, then $\mathit{head}(a) = \mathit{tail}(a) = \nu$.

In general, when $\mathcal{N}$ is not connected, $\mathcal{N}_{p+1}$ has as many nodes as there are components in $\mathcal{N}$.

Lemma 33. *If C is an auxiliary capacity constraint defined in the course of the construction of augmented networks induced by the non-augmented network $\mathcal{N} = (\mathbf{N}, \mathbf{A})$, where $\mathbf{A} = \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} \uplus \mathbf{A}_{\#}$, say C is:*

$$r \leqslant \sum A - \sum B \leqslant s,$$

where $A, B \subseteq \mathbf{A}$, for some $r, s \in \mathbb{R}$, then $A \cap \mathbf{A}_{\text{out}} = \varnothing$ and $B \cap \mathbf{A}_{\text{in}} = \varnothing$, i.e., input arcs/variables appear positively and output arcs/variables negatively in C.

Proof. Auxiliary capacity constraints are only introduced in point 8.2 of the construction. Consider the expressions φ_1 and φ_2 introduced in point 8.2. Both expressions satisfy the conclusion of the lemma, as does the substitution σ_k, because only the head but not the tail of an input arc (resp., the tail but not the head of an output arc) is defined. □

Lemma 34. *Let $\mathcal{N} = (\mathbf{N}, \mathbf{A})$ be a non-augmented network. For the last augmented network in the sequence of augmented networks induced by $\mathcal{N}$, namely, $\mathcal{N}_{p+1} = (\mathbf{N}_{p+1}, \mathbf{A}_{p+1}, \mathscr{E}_{p+1}, \mathscr{C}_{p+1}, \mathscr{C}^*_{p+1})$, we have:*

$$\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})\big) = \big[\mathsf{Poly}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})\big]_{\mathbf{A}_{\text{in,out}}}.$$

Proof. We start by carefully reviewing some notions regarding the decomposition of linear spaces and adapting them to our situation. Let $\mathbf{A}_{p+1} = \mathbf{A}_{\text{in,out}} \uplus \mathbf{A}'$ where:

$$\mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} = \{a_1, \ldots, a_m\} \quad \text{and} \quad \mathbf{A}' = \mathbf{A}_{p+1,\#} = \{a_{m+1}, \ldots, a_{m+n}\}.$$

We take $a_1, \ldots, a_{m+n}$, in this order, as denoting the axes of the $m+n$-dimensional space $\mathbb{R}^{m+n}$. We define the m-dimensional linear subspace S of $\mathbb{R}^{m+n}$ as follows:

$$S = \{\, \langle r_1, \ldots, r_m, \underbrace{0, \ldots, 0}_{n} \rangle \mid r_1, \ldots, r_m \in \mathbb{R} \,\},$$

which is just the projection of $\mathbb{R}^{m+n}$ on its first m coordinates. By definition, $\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})\big)$ is a polytope in the space $\mathbb{R}^m$, which is isomorphic to S. We can therefore view $\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})\big)$ – in this proof only[10] – as a polytope in the subspace S of $\mathbb{R}^{m+n}$. For every $h = \langle s_1, \ldots, s_n \rangle \in \mathbb{R}^n$, we also define the subset $S[h]$ of $\mathbb{R}^{m+n}$:

$$S[h] = \{\, \langle r_1, \ldots, r_m, s_1, \ldots, s_n \rangle \mid r_1, \ldots, r_m \in \mathbb{R} \,\}.$$

[10] $\mathbb{R}^m$ is often said, rather loosely, to be a linear subspace of $\mathbb{R}^{m+n}$, although there are different ways of embedding the first in the second. By identifying $\mathbb{R}^m$ with S, we choose a particular way of embedding $\mathbb{R}^m$ in $\mathbb{R}^{m+n}$.

$S[h]$ is what is usually called an *affine set* (or *flat* in some accounts), because it is a translate of the subspace S. The subspace S is none other than $S[\mathbf{0}]$ with $\mathbf{0} = \langle 0, \ldots, 0 \rangle \in \mathbb{R}^n$. For all $h_1, h_2 \in \mathbb{R}^n$, if $h_1 \neq h_2$, then $S[h_1]$ and $S[h_2]$ are parallel affine sets, because $S[h_1] \cap S[h_2] = \varnothing$. We can thus decompose $\mathbb{R}^{m+n}$ into an (infinite) union of affine sets, all parallel to the linear subspace S:

$$\mathbb{R}^{m+n} \;=\; \bigcup \big\{\, S[h] \;\big|\; h \in \mathbb{R}^n \,\big\}.$$

We can view $h \in \mathbb{R}^n$ as a function $h : \mathbf{A}' = \{a_{m+1}, \ldots, a_{m+n}\} \to \mathbb{R}$, with the obvious meaning when we apply h to $\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1}$; namely, we replace every arc/variable $a \in \mathbf{A}'$ occurring in $\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1}$ by the number $h(a)$ to obtain $h(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})$. In fact, $h(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1}) = \mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})$, because all the internal arcs in $\mathbf{A}'$ are self-loops by Lemma 32 and therefore do not occur in $\mathscr{E}_{p+1}$. By standard considerations of linear spaces, we have:

$$\mathsf{Poly}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1}) \;=\; \bigcup \big\{\, \mathsf{Poly}\big(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1})\big) \;\big|\; h : \mathbf{A}' \to \mathbb{R} \,\big\},$$

as well as, for every $h : \mathbf{A}' \to \mathbb{R}$, the equality:

$$\mathsf{Poly}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1}) \cap S[h] \;= \mathsf{Poly}\big(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1})\big).$$

By similar considerations, also reviewing procedure $\mathbf{PT}$ in Definition 22, we have:

$$\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})\big) \;=\; \bigcup \big\{\, \mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1}))\big) \;\big|\; h : \mathbf{A}' \to \mathbb{R} \,\big\}.$$

Hence, to complete the proof, it suffices to show that, for every $h : \mathbf{A}' \to \mathbb{R}$, the equation (\$) below holds:

$$(\$) \quad \mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1}))\big) \;=\; \big[\mathsf{Poly}\big(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1})\big)\big]_{\mathbf{A}_{\text{in,out}}}.$$

The arcs/variables in $\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1})$ are all in $\mathbf{A}_{\text{in,out}}$, because h maps every $a \in \mathbf{A}'$ to a number. $S[h]$ being a translate of S, we can view the polytope on the left-hand side of (\$) as contained in $S[h]$ instead of S, so that proving (\$) is equivalent to proving (\$\$) below for every $h : \mathbf{A}' \to \mathbb{R}$:

$$(\$\$) \quad \mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1}))\big) \;=\; \mathsf{Poly}\big(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1})\big).$$

Alternatively, we can view $\mathsf{Poly}\big(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1})\big)$ as contained in S instead of $S[h]$, justifying again the equivalence of (\$) and (\$\$).

To prove (\$\$), let $h = \langle s_1, \ldots, s_n \rangle$ be a fixed, but otherwise arbitrary, assignment of values to the internal arcs $\mathbf{A}' = \{a_{m+1}, \ldots, a_{m+n}\}$. By Definition 22 and the way procedure $\mathbf{PT}$ is set up, the following inclusion is immediate:

$$\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1}))\big) \;\supseteq\; \mathsf{Poly}\big(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1})\big).$$

The harder part is the opposite inclusion:

$$\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup \mathscr{C}^*_{p+1}))\big) \;\subseteq\; \mathsf{Poly}\big(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup \mathscr{C}^*_{p+1})\big).$$

The way we proceed is to show that, for every equation or constraint in $\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1})$, there is an equation or constraint in $\mathbf{PT}(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup h(\mathscr{C}^*_{p+1}))$ which is at least as restrictive. Let T be the typing $T = \mathbf{PT}\big(\mathscr{E}_{p+1} \cup h(\mathscr{C}_{p+1}) \cup \mathscr{C}^*_{p+1})\big)$. We consider each of three cases separately:

- E is an equation in $\mathscr{E}_{p+1}$,
- C is a capacity constraint in $h(\mathscr{C}_{p+1})$,

- C^* is an auxiliary capacity constraint in $h(\mathscr{C}^*_{p+1})$.

An equation E in $\mathscr{E}_{p+1}$ is of the form $\sum A = \sum B$ for some $A \subseteq \mathbf{A}_{\text{in}}$ and $B \subseteq \mathbf{A}_{\text{out}}$, with $|A| \geqslant 1$ and $|B| \geqslant 1$. The equation E gives rise to the type assignment $T(A \cup B) = [0,0]$, and the two are equally restrictive, because "$T(A \cup B) = [0,0]$" means $0 \leqslant \sum A - \sum B \leqslant 0$.

A capacity constraint C in $h(\mathscr{C}_{p+1})$ is of the form $\underline{\boldsymbol{c}}(a) \leqslant h(a) \leqslant \overline{\boldsymbol{c}}(a)$ for some $a \in \mathbf{A}_{p+1}$, for which there are two subcases:

- If $a \in \mathbf{A}_{\text{in,out}}$, then $h(a) = a$ and C is the constraint $\underline{\boldsymbol{c}}(a) \leqslant a \leqslant \overline{\boldsymbol{c}}(a)$. There is a corresponding type assignment $T(\{a\}) = [r_1, r_2]$ such that necessarily $\underline{\boldsymbol{c}}(a) \leqslant r_1$ and $r_2 \leqslant \overline{\boldsymbol{c}}(a)$, by the definition of procedure $\mathbf{PT}$.
- If $a \in \mathbf{A}'$, then $h(a)$ is a constant s and C is the constraint $\underline{\boldsymbol{c}}(a) \leqslant s \leqslant \overline{\boldsymbol{c}}(a)$. If it is indeed that $\underline{\boldsymbol{c}}(a) \leqslant s$ and $s \leqslant \overline{\boldsymbol{c}}(a)$, then C is omitted altogether from the computation of procedure $\mathbf{PT}$. If $\underline{\boldsymbol{c}}(a) > s$ or $s > \overline{\boldsymbol{c}}(a)$, then C cannot be satisfied and both sides of equation (\$\$) are $\varnothing$.

An auxiliary capacity constraint C^* in $h(\mathscr{C}^*_{p+1})$ is of the form, by Lemma 33:

$$s_1 \;\leqslant\; (\sum A - \sum B) + h(\sum A' - \sum B') \;\leqslant\; s_2.$$

where $A \subseteq \mathbf{A}_{\text{in}}$, $B \subseteq \mathbf{A}_{\text{out}}$, and $A' \cup B' \subseteq \mathbf{A}'$. Let $A \subseteq \mathbf{A}_{\text{in}}$ and $B \subseteq \mathbf{A}_{\text{out}}$ be sets of input/output arcs for which such an auxiliary constraint exists in $h(\mathscr{C}^*_{p+1})$. (It is not the case that, for *every* $A \subseteq \mathbf{A}_{\text{in}}$ and *every* $B \subseteq \mathbf{A}_{\text{out}}$, such an auxiliary constraint exists.) Define the numbers $\tilde{s}_1, \tilde{s}_2 \in \mathbb{R}$ as:

$$\tilde{s}_1 \;\coloneqq\; s_1 - h(\sum A' - \sum B') \quad \text{and} \quad \tilde{s}_2 \;\coloneqq\; s_2 - h(\sum A' - \sum B').$$

We can therefore write the preceding auxiliary constraint as:

$$\tilde{s}_1 \;\leqslant\; \sum A - \sum B \;\leqslant\; \tilde{s}_2.$$

For the same $A \subseteq \mathbf{A}_{\text{in}}$ and $B \subseteq \mathbf{A}_{\text{out}}$, the type assignment $T(A \cup B) = [r_1, r_2]$ is equivalent to a constraint:

$$r_1 \;\leqslant\; \sum A - \sum B \;\leqslant\; r_2.$$

(In contrast to $h(\mathscr{C}^*_{p+1})$, for *every* $A \subseteq \mathbf{A}_{\text{in}}$ and *every* $B \subseteq \mathbf{A}_{\text{out}}$, typing T computes such a type $[r_1, r_2]$.) By the definition of $\mathbf{PT}$ again, we necessarily have $[r_1, r_2] \subseteq [\tilde{s}_1, \tilde{s}_2]$. Hence, the type $T(A \cup B) = [r_1, r_2]$ is at least as restrictive as the auxiliary capacity constraint C^*. □

Proof of Theorem 26. Consider the sequence of augmented networks induced by the non-augmented network $\mathcal{N} = (\mathbf{N}, \mathbf{A})$. We have the following sequence of equalities:

$$\begin{aligned}
\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_1 \cup \mathscr{C}_1 \cup \mathscr{C}^*_1)\big) &= \mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})\big) && \text{by part 1 in Lemma 31,} \\
&= \big[\mathsf{Poly}(\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1})\big]_{\mathbf{A}_{\text{in,out}}} && \text{by Lemma 34,} \\
&= \big[\mathsf{Poly}(\mathscr{E}_1 \cup \mathscr{C}_1 \cup \mathscr{C}^*_1)\big]_{\mathbf{A}_{\text{in,out}}} && \text{by part 2 in Lemma 31.}
\end{aligned}$$

Hence, $\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E}_1 \cup \mathscr{C}_1 \cup \mathscr{C}^*_1)\big) = \big[\mathsf{Poly}(\mathscr{E}_1 \cup \mathscr{C}_1 \cup \mathscr{C}^*_1)\big]_{\mathbf{A}_{\text{in,out}}}$, which implies:

$$\mathsf{Poly}\big(\mathbf{PT}(\mathscr{E} \cup \mathscr{C})\big) = \big[\mathsf{Poly}(\mathscr{E} \cup \mathscr{C})\big]_{\mathbf{A}_{\text{in,out}}}$$

because $\mathscr{C}^*_1 = \varnothing$. But this is precisely the conclusion of Theorem 26, now expressed algebraically.

Remark 35. The proof of Theorem 26 above suggests two alternative procedures for inferring a principal typing for a given (non-augmented) flow network $\mathcal{N} = (\mathbf{N}, \mathbf{A}, \mathscr{E}, \mathscr{C})$ where, following the notation in this appendix, $|\mathbf{A}_{\#}| = p \geqslant 0$ and $|\mathbf{N}| = \ell \geqslant 1$. Call these two alternatives $\mathbf{PT}'$ and $\mathbf{PT}''$ to distinguish them from the original procedure $\mathbf{PT}$ in Definition 22.

The original $\mathbf{PT}$ works directly on $\mathscr{E} \cup \mathscr{C}$ and returns a typing $T = \mathbf{PT}(\mathscr{E} \cup \mathscr{C})$ for $\mathcal{N}$. Alternative procedure $\mathbf{PT}'$ works on the auxiliary capacity constraints of the last augmented network $\mathcal{N}_{p+1}$ in the sequence induced by $\mathcal{N}$. Alternative procedure $\mathbf{PT}''$ computes ℓ principal typings, one for each of the ℓ nodes in $\mathcal{N}$, and then combines them to produce a single principal typing for the entire network $\mathcal{N}$. More specifically below.

Procedure $\mathbf{PT}'$. We compute a principal typing T' for the given $\mathcal{N}$ in two stages:

1. Compute the sequence of augmented networks induced by $\mathcal{N}$ and let:

   $$\mathcal{N}_{p+1} = (\mathbf{N}_{p+1}, \mathbf{A}_{p+1}, \mathscr{E}_{p+1}, \mathscr{C}_{p+1}, \mathscr{C}^*_{p+1}).$$

   be the last network in the sequence where $\mathbf{A}_{p+1,\text{in,out}} = \mathbf{A}_{\text{in,out}}$.

2. For every $A \subseteq \mathbf{A}_{\text{in,out}}$, compute the minimum r_1 and maximum r_2 of the objective function:

   $$\theta_A \;=\; \sum\{\, a \mid a \in A \cap \mathbf{A}_{\text{in}} \,\} - \sum\{\, a \mid a \in A \cap \mathbf{A}_{\text{out}} \,\}$$

   relative to $\mathscr{E}_{p+1} \cup \mathscr{C}_{p+1} \cup \mathscr{C}^*_{p+1}$. Assign type $[r_1, r_2]$ to A.

By the analysis earlier in this appendix, typing T returned by $\mathbf{PT}$ and typing T' returned by $\mathbf{PT}'$ are equal. Observe that optimization using linear programming is only needed in stage 2.

Procedure $\mathbf{PT}''$. We compute a principal typing T'' for the given $\mathcal{N}$ in two stages:

1. Let $\mathbf{N} = \{\nu_1, \nu_2, \ldots, \nu_\ell\}$, the set of nodes in $\mathcal{N}$. Compute a principal typing T_i for each node ν_i, viewed as a single-node network.

2. Combine the one-node principal typings $\{T_1, T_2, \ldots, T_\ell\}$ using the methodology we develop elsewhere to obtain a principal typing T'' for the original $\mathcal{N}$. See [KM12].

In contrast to both $\mathbf{PT}$ and $\mathbf{PT}'$, we can carry out both stages of $\mathbf{PT}''$ without invoking any linear programming algorithm. We can compute the one-node principal typings $\{T_1, T_2, \ldots, T_\ell\}$ by inspection, and the operations in [KM12] do not invoke any linear-programming algorithm. However, to *prove* that T and T'' are equal, we need to show that, at every step of combining typings in $\{T_1, T_2, \ldots, T_\ell\}$ in stage 2, the resulting typing is equivalent to one obtained by linear-programming optimization, thus establishing at the end that T'' is equal to T (the latter is obtained by invoking an algorithm for linear programming).

There are complexity trade-offs between $\mathbf{PT}$, $\mathbf{PT}'$, and $\mathbf{PT}''$, which we do not examine in this report. □