# Flow Logic

Orna Kupferman and Gal Vardi

School of Computer Science and Engineering, The Hebrew University, Israel[*]

## Abstract

A flow network is a directed graph in which each edge has a capacity, bounding the amount of flow that can travel through it. Flow networks have attracted a lot of research in computer science. Indeed, many questions in numerous application areas can be reduced to questions about flow networks. This includes direct applications, namely a search for a maximal flow in networks, as well as less direct applications, like maximal matching or optimal scheduling. Many of these applications would benefit from a framework in which one can formally reason about properties of flow networks that go beyond their maximal flow.

We introduce *Flow Logics*: modal logics that treat flow functions as explicit first-order objects and enable the specification of rich properties of flow networks. The syntax of our logic BFL$^\star$ (Branching Flow Logic) is similar to the syntax of the temporal logic CTL$^\star$, except that atomic assertions may be *flow propositions*, like $> \gamma$ or $\geq \gamma$, for $\gamma \in \mathbb{N}$, which refer to the value of the flow in a vertex, and that first-order quantification can be applied both to paths and to flow functions. For example, the BFL$^\star$ formula $\mathcal{E}((\geq 100) \wedge AG(low \rightarrow (\leq 20))$ states that there is a legal flow function in which the flow is above $100$ and in all paths, the amount of flow that travels through vertices with low security is at most $20$.

We present an exhaustive study of the theoretical and practical aspects of BFL$^\star$, as well as extensions and fragments of it. Our extensions include flow quantifications that range over non-integral flow functions or over maximal flow functions, path quantification that ranges over paths along which non-zero flow travels, past operators, and first-order quantification of flow values. We focus on the *model-checking* problem and show that it is PSPACE-complete, as it is for CTL$^\star$. Handling of flow quantifiers, however, increases the complexity in terms of the network to $\mathrm{P}^{\mathrm{NP}}$, even for the LFL and BFL fragments, which are the flow-counterparts of LTL and CTL. We are still able to point to a useful fragment of BFL$^\star$ for which the model-checking problem can be solved in polynomial time.

**1998 ACM Subject Classification** F.4.1 Temporal Logic

**Keywords and phrases** Flow Network, Temporal Logic

**Digital Object Identifier** 10.4230/LIPIcs.CONCUR.2017.5

## 1 Introduction

A *flow network* is a directed graph in which each edge has a capacity, bounding the amount of flow that can travel through it. The amount of flow that enters a vertex equals the amount of flow that leaves it, unless the vertex is a *source*, which has only outgoing flow, or a *target*, which has only incoming flow. The fundamental *maximum-flow problem* gets as input a flow network and searches for a maximal flow from the source to the target [1, 2]. The problem was first formulated and solved in the 1950's [3, 4]. It has attracted much research on improved algorithms [5, 6, 7, 8] and applications [9].

The maximum-flow problem can be applied in many settings in which something travels along a network. This covers numerous application domains, including traffic in road or rail systems, fluids in pipes, currents in an electrical circuit, packets in a communication network, and many

[*] This research has received funding from the European Research Council under the EU's 7-th Framework Programme (FP7/2007-2013) / ERC grant agreement no 278410.

more [9]. Less obvious applications involve flow networks that are constructed in order to model settings with an abstract network, as in the case of scheduling with constraints [9] or elimination in partially completed tournaments [10]. In addition, several classical graph-theory problems can be reduced to the maximum-flow problem. This includes the problem of finding a maximum bipartite matching, minimum path cover, maximum edge-disjoint or vertex-disjoint path, and many more [1, 9]. Variants of the maximum-flow problem can accommodate further settings, like circulation problems [11], multiple source and target vertices, costs for unit flows, multiple commodities, and more [12].

All the above applications reduce the problem at hand to the problem of finding a maximal flow in a network. Often, however, one would like to reason about properties of flow networks that go beyond their maximal flow. This is especially true when the vertices or edges of the network attain information to which the properties can refer. For example, the vertices of a network may be labeled by their security level, and we may want to check whether all legal flow functions are such that the flow in every low-security vertex is at most 20, or check whether there is a flow function in which more than 100 units of flow reach the target and still the flow in every low-security vertex is at most 20. As another example, assume that each vertex in the network is labeled by the service provider that owns it, and we want to find a maximal flow under the constraint that flow travels through vertices owned by at most two providers.

The challenge of reasoning about properties of systems has been extensively studied in the context of formal verification. In particular, in *temporal-logic model checking* [13, 14], we check whether a system has a desired property by translating the system into a labeled state-transition graph, translating the property into a temporal-logic formula, and deciding whether the graph satisfies the formula. Model checking is one of the notable success stories of theoretical computer science, with exciting theoretical research that is being transformed into industrial applications [15, 16]. By viewing networks as labeled state-transition graphs, we can use existing model-checking algorithms and tools in order to reason about the structural properties of networks. We can check, for example, that every path from the source to the target eventually visits a *check-sum* vertex. Most interesting properties of flow networks, however, refer to flows and their values, and not just to the structural properties of the network. Traditional temporal logics do not support the specification and verification of such properties.

We introduce and study *Flow Logics*: modal logics that treat flow functions as explicit first-order objects and enable the specification of rich properties of flow networks. The syntax of our logic BFL$^\star$ (Branching Flow Logic) is similar to the syntax of the temporal logic CTL$^\star$, except that atomic assertions are built from both atomic propositions and *flow propositions*, like $> \gamma$ or $\geq \gamma$, for $\gamma \in \mathbb{N}$, which refer to the value of the flow in a vertex, and that first-order quantification can be applied both to paths and to flow functions. Thus, in addition to the path quantifiers $A$ ("for all paths") and $E$ ("there exists a path") that range over paths, states formulas may contain the flow quantifiers $\mathcal{A}$ ("for all flow functions") and $\mathcal{E}$ ("there exists a flow function"). For example, the BFL$^\star$ formula $\mathcal{E}((\geq 100) \wedge AG(low \rightarrow (\leq 20)))$ states the property discussed above, namely that there is a flow function in which the value of the flow is at least 100, and in all paths, the value of flow in vertices with low security is at most 20.

We study the theoretical aspects of BFL$^\star$ as well as extensions and fragments of it. We demonstrate their applications in reasoning about flow networks, and we examine the complexity of their model-checking problem. Below we survey briefly our results.

The traditional definition of flow considers *integral flow functions*: each edge is assigned a value in $\mathbb{N}$. Integral-flow functions arise naturally in settings in which the objects we transfer along the network cannot be partitioned into fractions, as is the case with cars, packets, and more. Sometimes, as in the case of liquids, flow can be partitioned arbitrarily. It is well-known, however, that maximum flow can be achieved by integral flows [3], thus algorithms consider integral flow even in settings in which flow can be partitioned arbitrarily. We show that, interestingly, in the richer setting of flow logic, restricting attention to integral flows may change the satisfaction value of formulas. Intuitively, it follows from the ability of a formula to specify properties that require unit flows to be partitioned. Accordingly, our semantics for BFL$^\star$ considers two types of flow quantification: one over integral flows and the second over non-integral ones.

Traditional branching temporal logics are insensitive to unwinding, in the sense that unwinding a system into a tree does not affect the satisfaction of CTL$^\star$ formulas. Indeed, a graph and its unwinding are *bisimilar* [17]. We prove that bisimulation is not a suitable equivalence relation for flow logics, which are sensitive to unwinding. One implication of this is that tree automata, which offer an effective framework for reasoning about branching temporal logics, cannot be easily used for reasoning about branching flow logics. Another implication is the usefulness of *past operators* in flow logic. We extend BFL$^\star$ also with such operators and study additional aspects of the expressive power of BFL$^\star$. In particular, we show that while the syntax of the linear fragment of BFL$^\star$, namely LFL (Linear Flow Logic), is similar to that of the linear temporal logic LTL, its semantics mixes the linear and branching views. Indeed, while LFL formulas describe paths in the network, the flow quantifiers in them refer to the network as a whole.

The flow propositions in BFL$^\star$ include constants. This makes it impossible to relate the flow in different vertices other than specifying all possible constants that satisfy the relation. Another extension for BFL$^\star$ that we consider is by *first-order quantification on flow values*. We also allow the logic to apply arithmetic operations on the values of quantified flow variables. For example, the formula $\mathcal{E}AG\forall x(x \rightarrow EX(\geq x \text{ div } 2))$, states that there is a flow in which all vertices have a successor that has at least half of their flow.

Other extensions we consider allow path quantifiers $A^+$ and $E^+$ that range over paths on which flow travels (rather than over all paths in the network), and allow flow quantifiers $\mathcal{A}^{max}$ and $\mathcal{E}^{max}$ that range over flow functions that attain the maximal flow. With such quantifiers we can express, for example, the property $\mathcal{E}^{max}(AX(A^+Xa \vee A^+Xb))$, stating that a maximal flow may be attained even if all the successors of the source direct their incoming flow only to vertices labeled $a$ or only to vertices labeled $b$. As demonstrated in Examples 5 and 6, such properties are useful in restricting matching and scheduling solutions in various settings.

We turn to examine the complexity of BFL$^\star$, its extensions, and its fragments. We show that algorithms for temporal-logic model-checking can be extended to handle flow logics, and that the complexity the BFL$^\star$ model-checking problems is PSPACE-complete. We show that PSPACE-hardness holds already for LFL formulas with no atomic propositions, namely when the specification only refers to the values of the flow along a path in the network. In practice, a network is typically much bigger than its specification, and its size is the computational bottleneck. In temporal-logic model checking, researchers have analyzed the *system complexity* of model-checking algorithms, namely the complexity in terms of the system, assuming the specification is of a fixed length. There, the system complexity of LTL and CTL$^\star$ model checking is NLOGSPACE-complete [18, 19]. We prove that, unfortunately, this is not the case for BFL$^\star$. That is, we prove that while the *network complexity* of the model-checking problem, namely the complexity in terms of the network, does not reach PSPACE, it does require polynomially many calls to an NP oracle. Essentially, each evaluation of a flow quantifier requires such a call, which increases the network complexity to $\Delta_2^P$.[1] We show that NP-hardness applies already for BFL – the flow-counterpart of CTL, for LFL, and for networks that are not labeled.

[1] The complexity class $\Delta_2^P$ includes all problems that can be solved by a deterministic polynomial-time Turing machine that has an oracle to a nondeterministic polynomial-time Turing machine, *a.k.a* $\mathrm{P}^{\mathrm{NP}}$.

We extend the algorithms to handle the various extensions of BFL$^\star$, and we show that they do not increase the complexity. The techniques for handling the extensions are, however, richer. We also consider the *flow-synthesis* problem. There, the checked formula contains a single existential flow quantifier and one has to return a flow function with which the formula is satisfied. We show that the complexities of the model-checking and the flow-synthesis problems coincide.

Since the network complexity is the computational bottleneck in model checking, the NP dependency in the size of the network motivates a definition of a feasible fragment of BFL$^\star$. Our fragment, *conjunctive-BFL*$^\star$ (CBFL$^\star$, for short), contains BFL$^\star$ formulas in which quantified flow functions are restricted in a conjunctive way. That is, when we "prune" a CBFL$^\star$ formula into requirements on the network, flow propositions are only conjunctively related. This enables the model-checking algorithm to search for quantified flow functions in a manner that is similar to a search for a maximal flow. More precisely, a maximal flow with lower and upper bounds on flow on vertices, which can be done in polynomial time. We show that many interesting properties can be specified in CBFL$^\star$. On a high-level view, it is interesting to compare the way CTL$^\star$ model checking is reduced to a sequence of reachability queries on a modified system – one that includes the restrictions imposed by the specification, and the way CBFL$^\star$ model checking is reduced to a sequence of maximal-flow queries in a modified network – one that includes the restrictions imposed by the specification.

**Related Work** There are three types of related works: (1) efforts to generalize the maximal-flow problem to richer settings, (2) extensions of temporal logics by new elements, in particular first-order quantification over new types, and (3) works on logical aspects of networks and their use in formal methods. Below we briefly survey them and their relation to our work.

As discussed early in this section, numerous extensions to the classical maximal-flow problems have been considered. In particular, some works that add constraints on the maximal flow, like capacities on vertices, or lower bounds on the flow along edges. Closest to flow logics are works that refer to labeled flow networks. For example, [20] considers flow networks in which edges are labeled, and the problem of finding a maximal flow with a minimum number of labels. Then, the maximal utilization problem of *capacitated automata* [21] amounts to finding maximal flow in a labeled flow network where flow is constrained to travel only along paths that belong to a given regular language. Our work suggests a formalism that embodies all these extensions, as well as a framework for formally reasoning about many more extensions and settings.

The competence of temporal-logic model checking initiated numerous extensions of temporal logics, aiming to capture richer settings. For example, *real-time* temporal logics include clocks with a real-time domain [22], *epistemic temporal logics* include knowledge operators [23], and *alternating temporal logics* include game modalities [24]. Closest to our work is *strategy logic* [25], where temporal logic is enriched by first-order quantification of strategies in a game. Beyond the theoretical interest in strategy logic, it was proven useful in synthesizing strategies in multi-agent systems and in the solution of rational synthesis [26].

Finally, network verification is an increasingly important topic in the context of protocol verification [27]. Tools that allow verifying properties of network protocols have been developed [28, 29]. These tools support verification of network protocols in the design phase as well as runtime verification [30]. Some of these tools use a query language called *Network Datalog* in order to specify network protocols [31]. Verification of *Software Defined Networks* has been studied widely, for example in [32, 33, 34]. Verification of safety properties in networks with finite-state middleboxes was studied in [35]. Network protocols describe forwarding policies for packets, and are thus related to specific flow functions. However, the way traffic is transmitted in these protocols does not correspond to the way flow travels in a flow network. Thus, properties verified in this line of work are different from these we can reason about with flow logic.

## 2 The Flow Logic BFL$^\star$

A *flow network* is $N = \langle AP, V, E, c, \rho, s, T\rangle$, where $AP$ is a set of atomic propositions, $V$ is a set of vertices, $s \in V$ is a source vertex, $T \subseteq V$ is a set of target vertices, $E \subseteq (V \setminus T) \times (V \setminus \{s\})$ is a set of directed edges, $c : E \to \mathbb{N}$ is a capacity function, assigning to each edge an integral amount of flow that the edge can transfer, and $\rho : V \to 2^{AP}$ assigns each vertex $v \in V$ to the set of atomic propositions that are valid in $v$. Note that no edge enters the source vertex or leaves a target vertex. We assume that all vertices $t \in T$ are reachable from $s$ and that each vertex has at least one target vertex reachable from it. For a vertex $u \in V$, let $E^u$ and $E_u$ be the sets of incoming and outgoing edges to and from $u$, respectively. That is, $E^u = (V \times \{u\}) \cap E$ and $E_u = (\{u\} \times V) \cap E$.

A *flow* is a function $f : E \to \mathbb{N}$ that describes how flow is directed in $N$. The capacity of an edge bounds the flow in it, thus for every edge $e \in E$, we have $f(e) \leq c(e)$. All incoming flow must exit a vertex, thus for every vertex $v \in V \setminus (\{s\} \cup T)$, we have $\sum_{e \in E^v} f(e) = \sum_{e \in E_v} f(e)$. We extend $f$ to vertices and use $f(v)$ to denote the flow that travels through $v$. Thus, for $v \in V \setminus (\{s\} \cup T)$, we define $f(v) = \sum_{e \in E^v} f(e) = \sum_{e \in E_v} f(e)$, for the source vertex $s$, we define $f(s) = \sum_{e \in E_s} f(e)$, and for a target vertex $t \in T$, we define $f(t) = \sum_{e \in E^t} f(e)$. Note that the preservation of flow in the internal vertices guarantees that $f(s) = \sum_{t \in T} f(t)$, which is the amount of flow that travels from $s$ to all the target vertices together. We say that a flow function $f$ is *maximal* if for every flow function $f'$, we have $f'(s) \leq f(s)$. A maximal flow function can be found in polynomial time [4]. The maximal flow for $N$ is then $f(s)$ for some maximal flow function $f$.

The logic BFL$^\star$ is a *Branching Flow Logic* that can specify properties of networks and flows in them. As in CTL$^\star$, there are two types of formulas in BFL$^\star$: *state formulas*, which describe vertices in a network, and *path formulas*, which describe paths. In addition to the operators in CTL$^\star$, the logic BFL$^\star$ has *flow propositions*, with which one can specify the flow in vertices, and *flow quantifiers*, with which one can quantify flow functions universally or existentially. When flow is not quantified, satisfaction is defined with respect to both a network and a flow function. Formally, given a set $AP$ of atomic propositions, a BFL$^\star$ state formula is one of the following:

**(S1)** An atomic proposition $p \in AP$.
**(S2)** A flow proposition $> \gamma$ or $\geq \gamma$, for an integer $\gamma \in \mathbb{N}$.
**(S3)** $\neg\varphi_1$ or $\varphi_1 \vee \varphi_2$, for BFL$^\star$ state formulas $\varphi_1$ and $\varphi_2$.
**(S4)** $A\psi$, for a BFL$^\star$ path formula $\psi$. $A$ is a path quantifier.
**(S5)** $\mathcal{A}\varphi$, for a BFL$^\star$ state formula $\varphi$. $\mathcal{A}$ is a flow quantifier.

A BFL$^\star$ path formula is one of the following:

**(P1)** A BFL$^\star$ state formula.
**(P2)** $\neg\psi_1$ or $\psi_1 \vee \psi_2$, for BFL$^\star$ path formulas $\psi_1$ and $\psi_2$.
**(P3)** $X\psi_1$ or $\psi_1 U \psi_2$, for BFL$^\star$ path formulas $\psi_1$ and $\psi_2$.

We say that a BFL$^\star$ formula $\varphi$ is *closed* if all flow propositions appear in the scope of a flow quantifier. The logic BFL$^\star$ consists of the set of closed BFL$^\star$ state formulas. We refer to state formula of the form $\mathcal{A}\varphi$ as a *flow state formula*.

The semantics of BFL$^\star$ is defined with respect to vertices in a flow network. Before we define the semantics, we need some more definitions and notations. Let $N = \langle AP, V, E, c, \rho, s, T\rangle$. For two vertices $u$ and $w$ in $V$, a finite sequence $\pi = v_0, v_1, \ldots, v_k \in V^*$ of vertices is a $(u,w)$*-path* in $N$ if $v_0 = u$, $v_k = w$, and $\langle v_i, v_{i+1}\rangle \in E$ for all $0 \leq i < k$. If $w \in T$, then $\pi$ is a *target* $u$*-path*.

State formulas are interpreted with respect to a vertex $v$ in $N$ and a flow function $f : E \to \mathbb{N}$. When the formula is closed, satisfaction is independent of the function $f$ and we omit it. We use $v, f \models \varphi$ to indicate that the vertex $v$ satisfies the state formula $\varphi$ when the flow function is $f$. The relation $\models$ is defined inductively as follows.

**(S1)** For an atomic proposition $p \in AP$, we have that $v, f \models p$ iff $p \in \rho(v)$.

**(S2)** For $\gamma \in \mathbb{N}$, we have $v, f \models > \gamma$ iff $f(v) > \gamma$ and $v, f \models \geq \gamma$ iff $f(v) \geq \gamma$.
**(S3a)** $v, f \models \neg\varphi_1$ iff $v, f \not\models \varphi_1$.
**(S3b)** $v, f \models \varphi_1 \vee \varphi_2$ iff $v, f \models \varphi_1$ or $v, f \models \varphi_2$.
**(S4)** $v, f \models A\psi$ iff for all target $v$-paths $\pi$, we have that $\pi, f \models \psi$.
**(S5)** $v, f \models \mathcal{A}\varphi$ iff for all flow functions $f'$, we have $v, f' \models \varphi$.

Path formulas are interpreted with respect to a finite path $\pi$ in $N$ and a flow function $f : E \to \mathbb{N}$. We use $\pi, f \models \varphi$ to indicate that the path $\pi$ satisfies the path-flow formula $\psi$ when the flow function is $f$. The relation $\models$ is defined inductively as follows. Let $\pi = v_0, v_1, \ldots, v_k$. For $0 \leq i \leq k$, we use $\pi^i$ to denote the suffix of $\pi$ that starts at $v_i$, thus $\pi^i = v_i, v_{i+1}, \ldots, v_k$.
**(P1)** For a state formula $\varphi$, we have that $\pi, f \models \varphi$ iff $v_0, f \models \varphi$.
**(P2a)** $\pi, f \models \neg\psi$ iff $\pi, f \not\models \psi$.
**(P2b)** $\pi, f \models \psi_1 \vee \psi_2$ iff $\pi, f \models \psi_1$ or $\pi, f \models \psi_2$.
**(P3a)** $\pi, f \models X\psi_1$ iff $k > 0$ and $\pi^1, f \models \psi_1$.
**(P3b)** $\pi, f \models \psi_1 U \psi_2$ iff there is $j \leq k$ such that $\pi^j, f \models \psi_2$, and for all $0 \leq i < j$, we have $\pi^i, f \models \psi_1$

For a network $N$ and a closed BFL$^\star$ formula $\varphi$, we say that $N$ satisfies $\varphi$, denoted $N \models \varphi$, iff $s \models \varphi$ (note that since $\varphi$ is closed, we do not specify a flow function).

Additional Boolean connectives and modal operators are defined from $\neg$, $\vee$, $X$, and $U$ in the usual manner; in particular, $F\psi = \mathbf{true}U\psi$ and $G\psi = \neg F \neg \psi$. We also define dual and abbreviated flow propositions: $< \gamma = \neg(\geq \gamma)$, $\leq \gamma = \neg(> \gamma)$, and $\gamma = (\leq \gamma) \wedge (\geq \gamma)$, a dual path quantifier: $E\psi = \neg A \neg \psi$, and a dual flow quantifier: $\mathcal{E}\varphi = \neg\mathcal{A}\neg\varphi$.

▶ **Example 1.** Consider a network $N$ in which target vertices are labeled by an atomic proposition *target*, and low-security vertices are labeled *red*. The BFL$^\star$ formula $\mathcal{E}EF(\mathit{target} \wedge 20)$ states that there is a flow in which 20 units reach a target vertex, and the BFL$^\star$ formula $\mathcal{A}((\geq 20) \to AX(\geq 4))$ states that in all flow functions in which the flow at the source is at least 20, all the successors must have flow of at least 4. Finally, $\mathcal{E}((\geq 100) \wedge AG(\mathit{red} \to (\leq 20)))$ states that there is a flow of at least 100 in which the flow in every low-security vertex is at most 20, whereas $\mathcal{A}((> 200) \to EF(\mathit{red} \wedge (> 20)))$ states that when the flow is above 200, then there must exist a low-security vertex in which the flow is above 20. As an example to a BFL$^\star$ formula with an alternating nesting of flow quantifiers, consider the formula $\mathcal{E}AG(< 10 \to \mathcal{A} < 15)$, stating that there is a flow such that wherever the flow is below 10, then in every flow it would be below 15. ◀

▶ Remark. Note that while the semantics of CTL$^\star$ and LTL is defined with respect to infinite trees and paths, path quantification in BFL$^\star$ ranges over finite paths. We are still going to use techniques and results known for CTL$^\star$ and LTL in our study. Indeed, for upper bounds, the transition to finite computations only makes the setting simpler. Also, lower-bound proofs for CTL$^\star$ and LTL are based on an encoding of finite runs of Turing machines, and apply also to finite paths.

Specifying finite paths, we have a choice between weak and strong semantics for the $X$ operator. In the weak semantics, the last vertex in a path satisfies $X\psi$, for all $\psi$. In particular, it is the only vertex that satisfies $X\mathbf{false}$. In the strong semantics, the last vertex does not satisfy $X\psi$, for all $\psi$. In particular, it does not satisfy $X\mathbf{true}$. We use the strong semantics.

## 3 Properties of BFL$^\star$

### 3.1 Integral vs. non-integral flow functions

Our semantics of BFL$^\star$ considers *integral flow functions*: vertices receive integral incoming flow and partition it to integral flows in the outgoing edges. Integral-flow functions arise naturally in

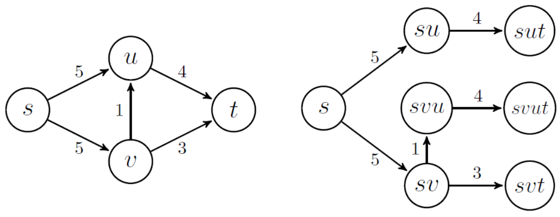


**Figure 1** The flow network $N$ and its unwinding $N_t$.

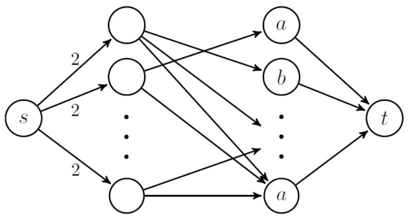
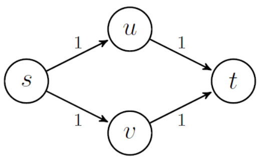


**Figure 2** Assigning workers to jobs.

settings in which the objects we transfer along the network cannot be partitioned into fractions, as is the case with cars, packets, and more. Sometimes, however, as in the case of liquids, flow can be partitioned arbitrarily. In the traditional maximum-flow problem, it is well known that the maximum flow can be achieved by integral flows [3]. We show that, interestingly, in the richer setting of flow logic, restricting attention to integral flows may change the satisfaction value of formulas.

▶ **Proposition 2.** *Allowing the quantified flow functions in BFL$^\star$ to get values in $\mathbb{R}$ changes its semantics.*

**Proof.** Consider the network on the right. The BFL$^\star$ formula $\varphi = \mathcal{E}(1 \wedge AX(>0))$ states that there is a flow function in which the flow that leaves the source is 1 and the flow of both its successors is strictly positive. It is easy to see that while no integral flow function satisfies the requirement in $\varphi$, a flow function in which 1 unit of flow in $s$ is partitioned between $u$ and $v$ does satisfy it. ◀



Proposition 2 suggests that quantification of flow functions that allow non-integral flows may be of interest. In Section 4.3 we discuss such an extension.

## 3.2 Sensitivity to unwinding

For a network $N = \langle AP, V, E, c, \rho, s, T\rangle$, let $N_t$ be the unwinding of $N$ into a tree. Formally, $N_t = \langle AP, V', E', \rho', s, T'\rangle$, where $V' \subseteq V^*$ is the smallest set such that $s \in V'$, and for all $w \cdot v \in V'$ with $w \in V^*$ and $v \in V \setminus T$, and all $u \in V$ such that $E(v, u)$, we have that $w \cdot v \cdot u \in V'$, with $\rho'(w \cdot v \cdot u) = \rho(u)$. Also, $\langle w \cdot v, w \cdot v \cdot u\rangle \in E'$, with $c'(\langle w \cdot v, w \cdot v \cdot u\rangle) = c(\langle v, u\rangle)$. Finally, $T' = V' \cap (V^* \cdot T)$. Note that $N_t$ may be infinite. Indeed, a cycle in $N$ induces infinitely many vertices in $N_t$.

The temporal logic CTL$^\star$ is insensitive to unwinding. Indeed, $N$ and $N_t$ are bisimilar, and for every CTL$^\star$ formula $\varphi$, we have $N \models \varphi$ iff $N_t \models \varphi$ [17]. We show that this is not the case for BFL$^\star$.

▶ **Proposition 3.** *The value of the maximal flow is sensitive to unwinding.*

**Proof.** Consider the network $N$ appearing in Figure 1, and its unwinding $N_t$ which appears in its right. It is easy to see that the value of the maximal flow in $N$ is 7 and the value of the maximal flow from $s$ to $T'$ in $N_t$ is 8. ◀

▶ **Corollary 4.** *The logic BFL$^\star$ is sensitive to unwinding.*

The sensitivity of BFL$^\star$ to unwinding suggests that extending BFL$^\star$ with past operators can increase its expressive power. In Section 6, we discuss such an extension.

## 4 Extensions and Fragments of BFL$^\star$

In this section we discuss useful extensions and variants of BFL$^\star$, as well as fragments of it. As we shall show in the sequel, while the extensions come with no computational price, their model checking requires additional techniques.

### 4.1 Positive path quantification

Consider a network $N = \langle AP, V, E, c, \rho, s, T\rangle$ and a flow function $f : E \to \mathbb{N}$. We say that a path $\pi = v_0, v_1, \ldots, v_k$ is positive if the flow along all the edges in $\pi$ is positive. Formally, $f(v_i, v_{i+1}) > 0$, for all $0 \leq i < k$. Note that it may be that $f(v_i) > 0$ for all $0 \leq i < k$ and still $\pi$ is not positive. It is sometimes desirable to restrict the range of path quantification to paths along which flow travels. This is the task of the *positive path quantifier* $A^+$, with the following semantics (dually, $E^+\psi = \neg A^+ \neg\psi$).

- $v, f \models A^+\psi$ iff for all positive target $v$-paths $\pi$, we have that $\pi, f \models \psi$.

▶ **Example 5.** Let $W$ be a set of workers and $J$ be a set of jobs. Each worker $w \in W$ can be assigned to perform jobs from a subset $J_w \subseteq J$. It is required to perform all jobs by assigning exactly one worker to each job and at most two jobs to each worker. This problem can be solved using a flow network. Indeed, a flow of $k$ units in the network described in Figure 2 corresponds to a legal assignment in which $k$ jobs are performed (in the figure, edges with no specified capacity have capacity 1). Now assume that some jobs should be processed in Location $a$ and the others in Location $b$. A worker can process jobs only in a single location, $a$ or $b$. By labeling the job-vertices by their location, the existence of a legal assignment in which $k$ jobs are processed can be expressed by the BFL$^\star$ formula $\mathcal{E}(k \wedge AX(A^+Xa \vee A^+Xb))$, which uses positive path quantification. ◀

### 4.2 Maximal flow quantification

It is sometimes desirable to restrict the range of flow quantification to maximal flow functions. This is the task of the *maximal-flow quantifier* $\mathcal{A}^{max}$, with the following semantics (dually, $\mathcal{E}^{max}\varphi = \neg\mathcal{A}^{max}\neg\varphi$).

- $v, f \models \mathcal{A}^{max}\varphi$ iff for all maximal-flow functions $f'$, we have that $v, f' \models \varphi$.

In a similar manner, it is sometimes helpful to relate to the maximal flow in the network. The max-flow constant $\gamma_{max} \in N$ maintains the value of the maximal flow from $s$ to $T$. We also allow arithmetic operations on $\gamma_{max}$.

▶ **Example 6.** Recall the job-assignment problem from Example 5. The BFL$^\star$ formula $\mathcal{E}^{max}(AX(A^+Xa\vee A^+Xb))$ states that the requirements about the locations do not reduce the number of jobs assigned without this requirement. Then, the formula $\mathcal{E}((\geq \gamma_{max} - 4) \wedge AX(A^+Xa \vee A^+Xb))$ states that the requirements about the locations may reduce the number of jobs performed by at most 4. ◀

### 4.3 Non-integral flow quantification

As discussed in Section 3.1, letting flow quantification range over non-integral flow functions may change the satisfaction value of a BFL$^\star$ formula. We extend BFL$^\star$ with a *non-integral flow quantifier* $\mathcal{A}^{\mathbb{R}}$, with the following semantics (dually, $\mathcal{E}^{\mathbb{R}}\varphi = \neg\mathcal{A}^{\mathbb{R}}\neg\varphi$).

- $v, f \models \mathcal{A}^{\mathbb{R}}\varphi$ iff for all flow functions $f' : E \to \mathbb{R}$, we have that $v, f' \models \varphi$.

### 4.4 Past operators

As discussed in Section 3.2, while temporal logics are insensitive to unwinding, this is not the case for BFL$^\star$. Intuitively, this follows from the fact that the flow in a vertex depends on the flow it gets from all its predecessors. This dependency suggests that an explicit reference to predecessors is useful, and motivates the extension of BFL$^\star$ by past operators.

Adding past to a branching logic, one can choose between a linear-past semantics – one in which past is unique (technically, the semantics is with respect to an unwinding of the network), and a branching-past semantics – one in which all the possible behaviors that lead to present are taken into an account (technically, the semantics is dual to that of future operators, and is defined with respect to the network) [36]. For flow logics, the branching-past approach is the suitable one, and is defined as follows.

For a path $\pi = v_0, v_1, \dots, v_k \in V^*$, a vertex $v \in V$, and index $0 \leq i \leq k$, we say that $\pi$ is a source-target $(v, i)$-path if $v_0 = s$, $v_i = v$, and $v_k \in T$. We add to BFL$^\star$ two past modal operators, $Y$ ("Yesterday") and $S$ ("Since"), and adjust the semantics as follows. Defining the semantics of logics that refer to the past, the semantics of path formulas is defined with respect to a path and an index in it. We use $\pi, i, f \models \psi$ to indicate that the path $\pi$ satisfies the path formula $\psi$ from position $i$ when the flow function is $f$. For state formulas, we adjust the semantics as follows.

**(S4)** $v, f \models A\psi$ iff for all source-target $(v, i)$-paths $\pi$, we have that $\pi, i, f \models \psi$.

Then, for path formulas, we have the following (the adjustment to refer to the index $i$ in all other modalities is similar).

- $\pi, i, f \models Y\psi_1$ iff $i > 0$ and $\pi, i-1, f \models \psi_1$.
- $\pi, i, f \models \psi_1 S \psi_2$ iff there is $0 \leq j < i$ such that $\pi, j, f \models \psi_2$, and for all $j+1 \leq l \leq i$, we have $\pi, l, f \models \psi_1$.

▶ **Example 7.** Recall the job-assignment problem from Example 5. Assume we want to apply the restriction about the location only to jobs that can be assigned only to workers with no car. Thus, if all the predecessors of a job-vertex are labeled by *car*, then this job can be served in either locations. Using past operators, we can specify this property by $\mathcal{E}(k \wedge AX(A^+X(a \vee AY\,car) \vee A^+X(b \vee AY\,car)))$. ◀

As proven in [36], adding past to CTL$^\star$ with a branching-past semantics strictly increases its expressive power. The same arguments can be used in order to show that BFL$^\star$ with past operators is strictly more expressive than BFL$^\star$.[2]

### 4.5 First-Order quantification on flow values

The flow propositions in BFL$^\star$ include constants. This makes it impossible to relate the flow in different vertices other than specifying all possible constants that satisfy the relation. In *BFL$^\star$ with quantified flow values* we add flow variables $X = \{x_1, \dots, x_n\}$ that can be quantified universally or existentially and specify such relations conveniently. We also allow the logic to apply arithmetic operations on the values of variables in $X$.

For a set of arithmetic operators $O$ (that is, $O$ may include $+, *$, etc.), let BFL$^\star(O)$ be BFL$^\star$ in which Rule S2 is extended to allow expressions with variables in $X$ constructed by operators in $O$, and we also allow quantification on the variables in $X$. Formally, we have the following:

[2] We note that the result from [36] does not immediately imply the addition of expressive power, as it is based on the fact that only CTL$^\star$ with past operators is sensitive to unwinding (and, as we prove in Section 3.2, BFL$^\star$ is sensitive to unwinding). Still, since the specific formula used in [36] in order to prove the sensitivity of CTL$^\star$ with past to unwinding does not refer to flow, it is easy to see that it has no equivalent BFL$^\star$ formula.

**(S2)** A flow proposition $> g(x_1,\dots,x_k)$ or $\geq\ g(x_1,\dots,x_k)$, where $x_1,\dots,x_k$ are variables in $X$ and $g$ is an expression obtained from $x_1,\dots,x_k$ by applying operators in $O$, possibly using constants in $\mathbb{N}$. We assume that $g : \mathbb{N}^k \to \mathbb{N}$. That is, $g$ leaves us in the domain $\mathbb{N}$.
**(S6)** $\forall x\varphi$, for $x \in X$ and a BFL$^\star(O)$ formula $\varphi$ in which $x$ is free.

For a BFL$^\star(O)$ formula $\varphi$ in which $x$ is a free variable, and a constant $\gamma \in \mathbb{N}$, let $\varphi[x \leftarrow \gamma]$ be the formula obtained by assigning $\gamma$ to $x$ and replacing expressions by their evaluation. Then, $v, f \models \forall x\varphi'$ iff for all $\gamma \in \mathbb{N}$, we have that $v, f \models \varphi'[x \leftarrow \gamma]$.

▶ **Example 8.** The logic BFL$^\star(\emptyset)$ includes the formula $\mathcal{E}AG\forall x((\mathit{split} \wedge x \wedge > 0) \to EX(> 0 \wedge < x))$, stating that there is a flow in which all vertices that are labeled *split* and with a positive flow $x$ have a successor in which the flow is positive but strictly smaller than $x$. Then, BFL$^\star(\mathrm{div})$ includes the formula $\mathcal{E}AG\forall x(x \to EX(\geq x \text{ div } 2))$, stating that there is a flow in which all vertices have a successor that has at least half of their flow.

Finally, BFL$^\star(+)$ includes the formula $\exists x\exists y\mathcal{E}^{max}AG(\neg(\mathit{source} \vee \mathit{target}) \to x \vee y \vee (x+y))$, stating that there are values $x$ and $y$, such that it is possible to attain the maximal flow by assigning to all vertices, except maybe source and target vertices, values in $\{x, y, x+y\}$. ◀

### 4.6 Fragments of BFL$^\star$

For the temporal logic CTL$^\star$, researchers have studied several fragments, most notably LTL and CTL. In this section we define interesting fragments of BFL$^\star$.

**Flow-CTL$^\star$ and Flow-LTL** The logics *Flow-CTL*$^\star$ and *Flow-LTL* are extensions of CTL$^\star$ and LTL in which atomic state formulas may be, in addition to $AP$s, also the flow propositions $> \gamma$ or $\geq \gamma$, for an integer $\gamma \in \mathbb{N}$. Thus, no quantification on flow is allowed, but atomic formulas may refer to flow. The semantics of Flow-CTL$^\star$ is defined with respect to a network and a flow function, and that of Flow-LTL is defined with respect to a path in a network and a flow function.

**Linear Flow Logic** The logic LFL is the fragment of BFL$^\star$ in which only one external universal path quantification is allowed. Thus, an LFL formula is a BFL$^\star$ formula of the form $A\psi$, where $\psi$ is generated without rule S4.

Note that while the temporal logic LTL is a "pure linear" logic, in the sense that satisfaction of an LTL formula in a computation of a system is independent of the structure of the system, the semantics of LFL mixes linear and branching semantics. Indeed, while all the paths in $N$ have to satisfy $\psi$, the context of the system is important. To see this, consider the LFL formula $\varphi = A\mathcal{A}((\geq 10) \to X(\geq 4))$. The formula states that in all paths, all flow functions in which the flow at the first vertex in the path is at least 10, are such that the flow at the second vertex in the path is at least 4. In order to evaluate the path formula $\mathcal{A}((\geq 10) \to X(\geq 4))$ in a path $\pi$ of a network $N$ we need to know the capacity of all the edges from the source of $N$, and not only the capacity of the first edge in $\pi$. For example, $\varphi$ is satisfied in networks in which there are two successors to the source, each connected by an edge with capacity 4, 5, or 6. Consider now the LFL formula $\varphi' = A\mathcal{E}(10 \wedge X(\geq 4))$. Note that $\varphi'$ is not equal to the BFL$^\star$ formula $\theta = \mathcal{E}(10 \wedge AX(\geq 4))$. Indeed, in the latter, the same flow function should satisfy the path formula $X(\geq 4)$ in all paths.

**No nesting of flow quantifiers** The logic BFL$^\star_1$ contains formulas that are Boolean combinations of formulas of the form $\mathcal{E}\varphi$ and $\mathcal{A}\varphi$, for a Flow-CTL$^\star$ formula $\varphi$. Of special interest are the following fragments of BFL$^\star_1$:

- $\exists$BFL$^\star_1$ and $\forall$BFL$^\star_1$, where formulas are of the form $\mathcal{E}\varphi$ and $\mathcal{A}\varphi$, respectively, for a Flow-CTL$^\star$ formula $\varphi$.
- $\exists$LFL$_1$ and $\forall$LFL$_1$, where formulas are of the form $\mathcal{E}A\psi$ and $\mathcal{A}A\psi$, respectively, for a Flow-LTL formula $\psi$, and LFL$_1$, where a formula is a Boolean combination of $\exists$LFL$_1$ and $\forall$LFL$_1$ formulas.

**Conjunctive-BFL$^\star$** The fragment Conjunctive-BFL$^\star$ (CBFL$^\star$, for short) contains BFL$^\star$ formulas whose flow state sub-formulas restrict the quantified flow in a conjunctive way. That is, when we "prune" a CBFL$^\star$ formula into requirements on the network, atomic flow propositions are only conjunctively related. This would have a computational significance in solving the model-checking problem.

Consider an $\exists$BFL$^\star_1$ formula $\varphi = \mathcal{E}\theta$. We say that an operator $g \in \{\vee, \wedge, E, A, X, F, G, U\}$ has a *positive polarity* in $\varphi$ if all the occurrences of $g$ in $\theta$ are in a scope of an even number of negations. Dually, $g$ has a *negative polarity* in $\varphi$ if all its occurrences in $\theta$ are in a scope of an odd number of negations.

The logic $\exists$CBFL$^\star_1$ is a fragment of $\exists$BFL$^\star_1$ in which the only operators with a positive polarity are $\wedge$, $A$, $X$, and $G$, and the only operators with a negative polarity are $\vee$, $E$, $X$, and $F$. Note that $U$ is not allowed, as its semantics involves both conjunctions and disjunctions. Note that by pushing negations inside, we make all operators of a positive polarity; that is, we are left only with $\wedge$, $A$, $X$, and $G$.

Then, since all requirements are universal and conjunctively related, we can push conjunctions outside so that path formulas do not have internal conjunctions – for example, transform $AX(\xi_1 \wedge \xi_2)$ into $AX\xi_1 \wedge AX\xi_2$, and can get rid of universal path quantification that is nested inside another universal path quantification – for example, transform $AXAX\xi_1$ into $AXX\xi$. Finally, since we use the strong semantics to $X$, we can replace formulas that have $X$ nested inside $G$ by **false**.

The logic $\forall$CBFL$^\star_1$ is the dual fragment of $\forall$BFL$^\star_1$. In other words, $\mathcal{A}\theta$ is in $\forall$CBFL$^\star_1$ iff $\mathcal{E}\neg\theta$ is in $\exists$CBFL$^\star_1$. The logic CBFL$^\star$ is then obtained by going up a hierarchy in which formulas of lower levels serve as atomic propositions in higher levels.

We now define the syntax of CBFL$^\star$ formally. For simplicity, we define it in a normal form, obtained by applying the rules described above. A CBFL$^\star_0$ formula is a Boolean assertion over $AP$. For $i \geq 0$, a CBFL$^\star_{i+1}$ formula is a Boolean assertion over CBFL$^\star_i$ formulas and formulas of the form $\mathcal{E}(A\psi_1 \wedge \cdots \wedge A\psi_n)$, where $\psi_j$ is of the form $X^{k_j}\xi_j$ or $X^{k_j}G\xi_j$, where $k_j \geq 0$ and $\xi_j$ is a CBFL$^\star_i$ formula or a flow proposition (that is, $> \gamma$, $< \gamma$, $\geq \gamma$, or $\leq \gamma$, for an integer $\gamma \in \mathbb{N}$). Then, a CBFL$^\star$ formula is a CBFL$^\star_i$ formula for some $i \geq 0$. Note that both $\exists$CBFL$^\star_1$ and $\forall$CBFL$^\star_1$ are contained in CBFL$^\star_1$.

▶ **Example 9.** Recall the job-assignment problem from Example 5. The CBFL$^\star_1$ formula $\mathcal{A}(< 10 \rightarrow EX \leq 0) \wedge \mathcal{E}(15 \wedge AX \geq 1)$ states that if less than 10 jobs are processed, then at least one worker is unemployed, but it is possible to process 15 jobs and let every worker process at least one job.

**BFL** The logic BFL is the fragment of BFL$^\star$ in which every modal operator $(X, U)$ is preceded by a path quantifier. That is, it is the flow counterpart of CTL.

## 5 Model Checking

In this section we study the model-checking problem for BFL$^\star$. The problem is decide, given a flow network $N$ and a BFL$^\star$ formula $\varphi$, whether $N \models \varphi$.

▶ **Theorem 10.** *BFL$^\star$ model checking is PSPACE-complete.*

**Proof.** Consider a network $N$ and a BFL$^\star$ formula $\varphi$. The idea behind our model-checking procedure is similar to the one that recursively employs LTL model checking in the process of CTL$^\star$ model checking [37]. Here, however, the setting is more complicated. Indeed, the path formulas in BFL$^\star$ are not "purely linear", as the flow quantification in them refers to flow in the (branching) network. In addition, while the search for witness paths is restricted to paths in the network, which can be guessed on-the-fly in the case of LTL, here we also search for witness flow functions, which have to be guessed in a global manner.

Let $\{\varphi_1, \ldots, \varphi_k\}$ be the set of flow state formulas in $\varphi$. Assume that $\varphi_1, \ldots, \varphi_k$ are ordered so that for all $1 \leq i \leq k$, all the subformulas of $\varphi_i$ have indices in $\{1, \ldots, i\}$. Our model-checking procedure labels $N$ by new atomic propositions $q_1, \ldots, q_k$ so that for all vertices $v$ and $1 \leq i \leq k$, we have that $v \models q_i$ iff $v \models \varphi_i$ (note that since $\varphi_i$ is closed, satisfaction is independent of a flow function).

Starting with $i = 1$, we model check $\varphi_i$, label $N$ with $q_i$, and replace the subformula $\varphi_i$ in $\varphi$ by $q_i$. Accordingly, when we handle $\varphi_i$, it is an $\exists$BFL$^\star_1$ or a $\forall$BFL$^\star_1$ formula. That is, it is of the form $\mathcal{E}\xi$ or $\mathcal{A}\xi$, for a Flow-CTL$^\star$ formula $\xi$. Assume that $\varphi_i = \mathcal{E}\xi$. We guess a flow function $f : E \to \mathbb{N}$, and perform CTL$^\star$ model-checking on $\xi$, evaluating the flow propositions in $\xi$ according to $f$. Since guessing $f$ requires polynomial space, and CTL$^\star$ model checking is in PSPACE, so is handling of $\varphi_i$ and of all the subformulas.

Hardness in PSPACE follows from the hardness of CTL$^\star$ model checking [38]. ◀

Since BFL$^\star$ contains CTL$^\star$, the lower bound in Theorem 10 is immediate. One may wonder whether reasoning about flow networks without atomic propositions, namely when we specify properties of flow only, is simpler. Theorem 11 below shows that this is not the case. Essentially, the proof follows from our ability to encode assignments to atomic propositions by values of flow.

▶ **Theorem 11.** *BFL$^\star$ model checking is PSPACE-complete already for $\forall$LFL$_1$ formulas without atomic propositions.*

We also note that when the given formula is in BFL, we cannot avoid the need to guess a flow function, yet once the flow function is guessed, we can verify it in polynomial time. Accordingly, the model-checking problem for BFL is in $\mathrm{P}^{\mathrm{NP}}$. We discuss this point further below.

In practice, a network is typically much bigger than its specification, and its size is the computational bottleneck. In temporal-logic model checking, researchers have analyzed the *system complexity* of model-checking algorithms, namely the complexity in terms of the system, assuming the specification is of a fixed length. There, the system complexity of LTL and CTL$^\star$ is NLOGSPACE-complete [18, 19]. We prove that, unfortunately, this is not the case of BFL$^\star$. That is, we prove that while the *network complexity* of the model-checking problem, namely the complexity in terms of the network, does not reach PSPACE, it does require polynomially many calls to an NP oracle. Essentially, each evaluation of a flow quantifier requires such a call. Formally, we have the following.

▶ **Theorem 12.** *The network complexity of BFL$^\star$ is in $\Delta_2^P$ (namely, in $\mathrm{P}^{\mathrm{NP}}$).*

**Proof.** Fixing the length of the formula in the algorithm described in the proof of Theorem 10, we get that $k$ is fixed, and so is the length of each subformula $\varphi_i$. Thus, evaluation of $\varphi_i$ involves a guess of a flow and then model checking of a fixed size Flow-CTL$^\star$ formula, which can be done in time polynomial in the size of the network. Hence, the algorithm from Theorem 10 combined with an NP oracle gives the required network complexity. ◀

While finding the exact network complexity of model checking BFL$^\star$ and its fragments is interesting from a complexity-theoretical point of view, it does not contribute much to our story. Here, we prove NP and co-NP hardness holds already for very restricted fragments. As good news, in Section 7 we point that for the conjunctive fragment, model checking can be performed in polynomial time.[3]

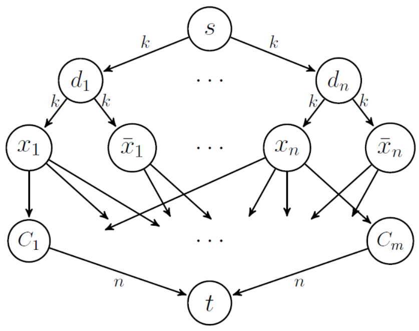
[3] A possible tightening of our analysis is via the complexity class BH, which is based on a Boolean hierarchy over NP. Essentially, it is the smallest class that contains NP and is closed under union, intersection, and complement.

▶ **Theorem 13.** *The network complexity of $\exists$BFL$^\star_1$ and $\forall$BFL$^\star_1$ is NP-complete and co-NP-complete, respectively. Hardness applies already to $\exists$LFL$_1$ and $\forall$LFL$_1$ without atomic propositions, and to BFL.*

**Proof.** For the upper bound, it is easy to see that one step in the algorithm described in the proof of Theorem 10 (that is, evaluating $\varphi_i$ once all its flow state subformulas have been evaluated), when applied to $\varphi_i$ of a fixed length is in NP for $\varphi_i$ of the form $\mathcal{E}\xi$ and in co-NP for $\varphi_i$ of the form $\mathcal{A}\xi$.

For the lower bound, we prove NP-hardness for $\exists$LFL$_1$. Co-NP-hardness for $\forall$LFL$_1$ follows by dualization. We describe a reduction from CNF-SAT. Let $\theta = C_1 \wedge \ldots \wedge C_m$ be a CNF formula over the variables $x_1 \ldots x_n$. We assume that every literal in $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$ appears exactly in $k$ clauses in $\theta$. Indeed, every CNF formula can be converted to such a formula in polynomial time and with a polynomial blowup.

We construct a flow network $N$ and an $\exists$LFL$_1$ formula $\mathcal{E}A\psi$ such that $\theta$ is satisfiable iff $N \models \mathcal{E}A\psi$. The network $N$ is constructed as demonstrated on the right Let $Z = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. For a literal $z \in Z$ and a clause $C_i$, the network $N$ contains an edge $\langle z, C_i\rangle$ iff the clause $C_i$ contains the literal $z$. Thus, each vertex in $Z$ has exactly $k$ outgoing edges. The capacity of each of these edges is 1. The flow-LTL formula $\psi = kn \wedge XX(k \vee 0) \wedge XXX(\geq 1)$. In Appendix A.2, we prove that $\theta$ is satisfiable iff $N \models \mathcal{E}A\psi$.



Finally, note that $\psi$ does not contain atomic propositions. Also, the same proof holds with the BFL formula $\psi = kn \wedge AXEXk \wedge AXAXAX(\geq 1)$. ◀

## 5.1 Flow synthesis

In the flow-synthesis problem, we are given a network $N$ and an $\exists$BFL$^\star_1$ formula $\mathcal{E}\varphi$, and we have to return a flow function $f$ with which $\varphi$ is satisfied in $N$, or declare that no such function exists. The problem is clearly at least as hard as CTL$^\star$ model checking. Also, by guessing $f$, its complexity does not go beyond CTL$^\star$ model-checking complexity. The network complexity of the problem coincides with that of $\exists$BFL$^\star_1$ model checking. Thus, we have the following.

▶ **Theorem 14.** *The flow-synthesis problem for $\exists$BFL$^\star_1$ is PSPACE-complete, and its network complexity is NP-complete.*

## 6 Model Checking Extensions of BFL$^\star$

In Section 4, we define several extensions of BFL$^\star$. In this section we study the model-checking complexity for each of the extensions, and show that they do not require an increase in the complexity. The techniques for handling them are, however, richer: For positive path quantification, we have to refine the network and add a path-predicate that specifies positive flow, in a similar way fairness is handled in temporal logics. For maximal-flow quantification, we have to augment the model-checking algorithm by calls to a procedure that finds the maximal flow. For non-integral flow quantification, we have to reduce the model-checking problem to a solution of a linear-programming

The levels of the hierarchy start with BH$_1$ = NP, and each level adds internal intersections as well as intersection with a co-NP (even levels) or an NP (odd levels) language [39]. BH is contained in $\Delta^P_2$. It is not hard to prove that the network complexity of the fragment of BFL$^\star_1$ that contains at most $k$ flow quantifiers is in BH$_{k+1}\cap$ co-BH$_{k+1}$. Indeed, the latter contain problems that are decidable in polynomial time with $k$ parallel queries to an NP oracle [40]. A BH$_k$ lower bound can also be shown.

system. For past operators, we have to extend the model-checking procedure for CTL$^\star$ with branching past. Finally, for first-order quantification over flow values, we have to first bound the range of relevant values, and then apply model checking to all relevant values.

**Positive path quantification** Given a network $N$, it is easy to generate a network $N'$ in which we add a vertex in the middle of each edge, and in which the positivity of paths correspond to positive flow in the new intermediate vertices. Formally, assuming that we label the new intermediate vertices by an atomic proposition $edge$, then the BFL$^\star$ path formula $\xi_{positive} = G(edge \to > 0)$ characterizes positive paths, and replacing a state formula $A\psi$ by the formula $A(\xi_{positive} \to \psi)$ restricts the range of path quantification to positive paths. Now, given a BFL$^\star$ formula $\varphi$, it is easy to generate a BFL$^\star$ formula $\varphi'$ such that $N \models \varphi$ iff $N' \models \varphi'$. Indeed, we only have to (recursively) modify path formulas so that vertices labeled $edge$ are ignored: $X\xi$ is replaced by $XX\xi$, and $\xi_1 U \xi_2$ is replaced by $(\xi_1 \vee edge)U(\xi_2 \wedge \neg edge)$. Hence, the complexity of model-checking is similar to BFL$^\star$.

**Maximal flow quantification** The maximal flow $\gamma_{max}$ in a flow network can be found in polynomial time. Our model-checking algorithm for BFL$^\star$ described in the proof of Theorem 10 handles each flow state subformula $\mathcal{E}\varphi$ by guessing a flow function $f : E \to \mathbb{N}$ with which the Flow-CTL$^\star$ formula $\varphi$ holds. For an $\mathcal{E}^{max}$ quantifier, we can guess only flow functions for which the flow leaving the source vertex is $\gamma_{max}$. In addition, after calculating the maximal flow, we can substitute $\gamma_{max}$, in formulas that refer to it, by its value. Hence, the complexity of model-checking is similar to that of BFL$^\star$.

**Non-integral flow quantification** Recall that our BFL$^\star$ model-checking algorithm handles each flow state subformula $\mathcal{E}\varphi$ by guessing a flow function $f : E \to \mathbb{N}$ with which the Flow-CTL$^\star$ formula holds. Moving to non-integral flow functions, the guessed function $f$ should be $f : E \to \mathbb{R}$, where we cannot bound the size or range of guesses.

Accordingly, in the non-integral case, we guess, for every vertex $v \in V$, an assignment to the flow propositions that appear in $\varphi$. Then, we perform two checks. First, that $\varphi$ is satisfied with the guessed assignment – this is done by CTL$^\star$ model checking, as in the case of integral flows. Second, that there is a non-integral flow function that satisfies the flow constraints that appear in the vertices. This can be done in polynomial time by solving a system of inequalities [41] (see Lemma 17 for the details in the case of vertex-constrained integral flow functions). Thus, as in the integral case, handling each flow state formula $\mathcal{E}\varphi$ can be done in PSPACE, and so is the complexity of the entire algorithm.

**Past operators** Recall that our algorithm reduces BFL$^\star$ model checking to a sequence of calls to a CTL$^\star$ model-checking procedure. Starting with a BFL$^\star$ formula with past operators, the required calls are to a model-checking procedure for CTL$^\star$ with past. By [36], model checking CTL$^\star$ with branching past is PSPACE-complete, and thus so is the complexity of our algorithm.

**First-Order quantification on flow values** For a flow network $N = \langle AP, V, E, c, \rho, s, T\rangle$, let $C_N = 1 + \Sigma_{e\in E} c(e)$. Thus, for every flow function $f$ for $N$ and for every vertex $v \in V$, we have $f(v) < C_N$. We claim that when we reason about BFL$^\star$ formulas with quantified flow values, we can restrict attention to values in $\{0, 1, \ldots, C_N\}$.

▶ **Lemma 15.** *Let $N$ be a flow network and let $\theta = \forall x_1 \varphi$ be a BFL$^\star(\{+,*\})$ formula over the variables $X = \{x_1, \ldots, x_n\}$, and without free variables. Then, $N \models \theta$ iff $N \models \varphi[x_1 \leftarrow \gamma]$, for every $0 \le \gamma \le C_N$.*

▶ **Theorem 16.** *Model-checking for BFL$^\star(\{+,*\})$ is PSPACE-complete.*

**Proof.** The lower bound follows from the PSPACE-hardness of BFL$^\star$, which is contained in BFL$^\star(\{+,*\})$.

We prove the upper bound. Let $N$ be a flow network, let $\varphi$ be a BFL$^\star(\{+,*\})$ formula, and let $\varphi_1,\ldots,\varphi_k$ be the state subformulas of $\varphi$ that start with an outermost $\mathcal{A}$ or $\forall$ quantifier; that is, $\varphi_i$ is not in the scope of an outer quantifier. For every vertex $v$ in $N$ and every subformula $\varphi_i$, we check whether $v \models \varphi_i$ and label $v$ by a fresh atomic proposition $q_i$ that maintains the satisfaction of $\varphi_i$. By replacing $\varphi_i$ by $q_i$, we replace $\varphi$ by a CTL$^\star$ formula, we can model-check in PSPACE. For a vertex $v$ in $N$, we show how to check whether $v \models \varphi_i$.

Assume first that $\varphi_i = \mathcal{A}\theta$. For every flow function $f$, we need to check whether $v, f \models \theta$. Given a flow function $f$, for every vertex $u$ and every flow proposition that is not quantified in $\theta$ we label $u$ with a fresh atomic proposition that maintains the satisfaction of the flow propositions in $u$. We also replace these flow propositions in $\theta$ by the fresh atomic propositions and denote the resulting formula by $\theta'$. Then, we check whether $v \models \theta'$ by recursively calling this algorithm.

Assume now that $\varphi_i = \forall x\theta$. By Lemma 15, we need to check whether for every $0 \le \gamma \le C_N$, we have $v \models \theta[x \leftarrow \gamma]$. We do this by recursively calling this algorithm. ◀

## 7 A Polynomial Fragment

In this section we show that the model-checking problem for CBFL$^\star$ (see Section 4.6) can be solved in polynomial time.

Our model-checking algorithm reduces the evaluation of a CBFL$^\star$ formula into a sequence of solutions to the *vertex-constrained flow problem*. In this problem, we are given a flow network $N = \langle AP, V, E, c, \rho, s, T\rangle$ in which each vertex $v \in V$ is attributed by a range $[\gamma_l, \gamma_u] \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$. The problem is to decide whether there is a flow function $f : E \to \mathbb{R}$ such that for all vertices $v \in V$, we have $\gamma_l \le f(v) \le \gamma_u$.

▶ **Lemma 17.** *The vertex-constrained flow problem can be solved in polynomial time. If there is a solution that is a non-integral flow function, then there is also a solution that is an integral flow function, and the algorithm returns such a solution.*

▶ **Theorem 18.** *CBFL$^\star$ model checking can be solved in polynomial time.*

**Proof.** Let $N = \langle AP, V, E, c, \rho, s, T\rangle$, and consider a CBFL$^\star$ formula $\varphi$. If $\varphi$ is in CBFL$^\star_0$, we can clearly label in linear time all the vertices in $N$ by a fresh atomic proposition $p_\varphi$ that maintains the satisfaction of $\varphi$. That is, in all vertices $v \in V$, we have that $p_\varphi \in \rho(v)$ iff $v \models \varphi$. Otherwise, $\varphi$ is a CBFL$^\star_{i+1}$ formula for some $i \ge 0$. We show how, assuming that the vertices of $N$ are labeled by atomic propositions that maintain satisfaction of the subformulas of $\varphi$ that are CBFL$^\star_i$ formulas, we can label them, in polynomial time, by a fresh atomic proposition that maintains the satisfaction of $\varphi$.

Recall that $\varphi$ is a Boolean assertion over CBFL$^\star_i$ formulas and flow formulas of the form $\mathcal{E}(A\psi_1 \wedge \cdots \wedge A\psi_n)$, where each $\psi_j$ is of the form $X^{k_j}\xi_j$ or $X^{k_j}G\xi_j$, where $k_j \ge 0$ and $\xi_j$ is a CBFL$^\star_i$ formula or a flow proposition (that is, $> \gamma$, $< \gamma$, $\ge \gamma$, or $\le \gamma$, for an integer $\gamma \in \mathbb{N}$).

Since CBFL$^\star_i$ subformulas have already been evaluated, we describe how to evaluate subformulas of the form $\theta = \mathcal{E}(A\psi_1 \wedge \cdots \wedge A\psi_n)$. Intuitively, since the formulas in $\theta$ include no disjunctions, they impose constraints on the vertices of $N$ in a deterministic manner. These constraints can be checked in polynomial time by solving a vertex-constrained flow problem. Recall that for each $1 \le j \le n$, the formula $\psi_j$ is of the form $X^{k_j}\xi_j$ or $X^{k_j}G\xi_j$, for some $k_j \ge 0$, and a CBFL$^\star_i$ formula or a flow proposition $\xi_j$. In order to evaluate $\theta$ in a vertex $v \in V$, we proceed as follows. For each $1 \le j \le n$, the formula $\xi_j$ imposes either a Boolean constraint (in case $\xi_j$ is a CBFL$^\star_i$ formula) or a flow constraint (in case $\xi_j$ is a flow proposition) on a finite subset $V^v_j$ of $V$. Indeed, if $\psi_j = X^{k_j}\xi_j$, then $V^v_j$ includes all the vertices reachable from $v$ by a path of length $k_j$, and if $\psi_j = X^{k_j}G\xi_j$, then $V^v_j$ includes all the vertices reachable from $v$ by a path of length at least $k_j$. We attribute each vertex by the constraints imposes on it by all the conjuncts in $\theta$. If one of the Boolean constrains does not hold, then $\theta$ does not hold in $v$. Otherwise, we obtain a set of flow constraints for each vertex in $V$. For example, if $\theta = \mathcal{E}(AXXp \wedge AXX > 5 \wedge AG \leq 8)$, then in order to check whether $\theta$ holds in $s$, we assign the flow constraint $\leq 8$ to all the vertices reachable from $s$, and assign the flow constraint $> 5$ to all the successors of the successors of $s$. If one of these successors of successors does not satisfy $p$, we can skip the check for a flow and conclude that $s$ does not satisfy $\theta$. Otherwise, we search for such a flow, as described below.

The flow constrains for a vertex induce a closed, open, or half-closed range. The upper bound in the range may be infinity. For example, the constrains $> 6$, $< 10$, $\leq 8$ induce the half-closed range $(6, 8]$. Note that it may be that the induced range is empty. For example, the constraints $\leq 6$ and $> 8$ induce an empty range. Then, $\theta$ does not hold in $v$. Since we are interested in integral flows, we can convert all strict bounds to non-strict ones. For example, the range $(6, 8]$ can be converted to $[7, 8]$. Note that since we are interested in integral flow, a non-empty open range may not be satisfiable, and we refer to it as an empty range. For example, the range $(6, 7)$ is empty. Hence, the satisfaction of $\theta$ in $v$ is reduced to an instance of the vertex-constrained flow problem. By Lemma 17, deciding whether there is a flow function that satisfies the constraints can be solved in polynomial time. ◀

▶ Remark. Note that the same algorithm can be applied when we consider non-integral flow functions, namely in CBFL$^\star$ with the $\mathcal{A}^{\mathbb{R}}$ flow quantifier. There, the induced vertex-constrained flow problem may include open boundaries. The solution need not be integral, but can be found in polynomial time by solving a system of inequalities [41].

## References

1 T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

2 A. Goldberg, É. Tardos, and R. Tarjan, "Network flow algorithms," DTIC Document, Tech. Rep., 1989.

3 L. Ford and D. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, no. 3, pp. 399–404, 1956.

4 ——, *Flows in networks*. Princeton Univ. Press, Princeton, 1962.

5 J. Edmonds and R. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, 1972.

6 E. Dinic, "Algorithm for solution of a problem of maximum flow in a network with power estimation," *Soviet Math. Dokl*, vol. 11, no. 5, pp. 1277–1280, 1970, english translation by RF. Rinehart.

7 A. Goldberg and R. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM*, vol. 35, no. 4, pp. 921–940, 1988.

8 A. Madry, "Computing maximum flow with augmenting electrical flows," in *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*. IEEE, 2016, pp. 593–602.

9 R. Ahuja, T. Magnanti, and J. Orlin, *Network flows: Theory, algorithms, and applications*. Prentice Hall Englewood Cliffs, 1993.

10 B. Schwartz, "Possible winners in partially completed tournaments," *SIAM Review*, vol. 8, no. 3, pp. 302–308, 1966.

11 É. Tardos, "A strongly polynomial minimum cost circulation algorithm," *Combinatorica*, vol. 5, no. 3, pp. 247–255, 1985.

12 S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problems," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 691–703, 1976.

13 E. Clarke and E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Proc. Workshop on Logic of Programs*, ser. Lecture Notes in Computer Science, vol. 131. Springer, 1981, pp. 52–71.

**14** J. Queille and J. Sifakis, "Specification and verification of concurrent systems in Cesar," in *Proc. 9th ACM Symp. on Principles of Programming Languages*, ser. Lecture Notes in Computer Science, vol. 137. Springer, 1982, pp. 337–351.

**15** E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

**16** E. Clarke, T. Henzinger, and H. Veith, *Handbook of Model Checking*. Elsvier, 2016, forthcoming.

**17** R. Milner, "An algebraic definition of simulation between programs," in *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*. British Computer Society, 1971, pp. 481–489.

**18** O. Lichtenstein and A. Pnueli, "Checking that finite state concurrent programs satisfy their linear specification," in *Proc. 12th ACM Symp. on Principles of Programming Languages*, 1985, pp. 97–107.

**19** O. Kupferman, M. Vardi, and P. Wolper, "An automata-theoretic approach to branching-time model checking," *Journal of the ACM*, vol. 47, no. 2, pp. 312–360, 2000.

**20** D. Granata, R. Cerulli, M. Scutellá, and A. Raiconi, "Maximum flow problems and an NP-complete variant on edge-labeled graphs," in *Handbook of Combinatorial Optimization*. Springer, 2013, pp. 1913–1948.

**21** O. Kupferman and T. Tamir, "Properties and utilization of capacitated automata," in *Proc. 34th Conf. on Foundations of Software Technology and Theoretical Computer Science*, ser. LIPIcs, vol. 29. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2014, pp. 33–44.

**22** R. Alur and T. Henzinger, "A really temporal logic," *Journal of the ACM*, vol. 41, no. 1, pp. 181–204, 1994.

**23** J. Halpern and Y. Moses, "Knowledge and common knowledge in a distributed environment," *Journal of the ACM*, vol. 37, no. 3, pp. 549–587, 1990.

**24** R. Alur, T. Henzinger, and O. Kupferman, "Alternating-time temporal logic," *Journal of the ACM*, vol. 49, no. 5, pp. 672–713, 2002.

**25** K. Chatterjee, T. A. Henzinger, and N. Piterman, "Strategy logic," in *Proc. 18th Int. Conf. on Concurrency Theory*, 2007, pp. 59–73.

**26** D. Fisman, O. Kupferman, and Y. Lustig, "Rational synthesis," in *Proc. 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, vol. 6015. Springer, 2010, pp. 190–204.

**27** P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 113–126.

**28** A. Wang, P. Basu, B. Loo, and O. Sokolsky, "Declarative network verification," in *International Symposium on Practical Aspects of Declarative Languages*. Springer, 2009, pp. 61–75.

**29** N. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese, "Checking beliefs in dynamic networks," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 499–512.

**30** P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 99–111.

**31** B. Loo, T. Condie, M. Garofalakis, D. Gay, J. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica, "Declarative networking: language, execution and optimization," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 97–108.

**32** A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. Godfrey, "Veriflow: verifying network-wide invariants in real time," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 15–27.

**33** M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A soft way for openflow switch interoperability testing," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 265–276.

34 M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A nice way to test openflow applications," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 127–140.

35 Y. Velner, K. Alpernas, A. Panda, A. Rabinovich, M. Sagiv, S. Shenker, and S. Shoham, "Some complexity results for stateful network verification," in *Proc. 22nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2016, pp. 811–830.

36 O. Kupferman, A. Pnueli, and M. Vardi, "Once and forall," *Journal of Computer and Systems Science*, vol. 78, no. 3, pp. 981–996, 2012.

37 E. Emerson and C.-L. Lei, "Modalities for model checking: Branching time logic strikes back," *Science of Computer Programming*, vol. 8, pp. 275–306, 1987.

38 A. Sistla and E. Clarke, "The complexity of propositional linear temporal logic," *Journal of the ACM*, vol. 32, pp. 733–749, 1985.

39 G. Wechsung, "On the boolean closure of np," in *Proceedings of the International Conference on Fundamentals of Computation Theory*, ser. Lecture Notes in Computer Science, vol. 199. Springer, 1985, pp. 485–493.

40 R. Beigel, "Bounded queries to sat and the boolean hierarchy," *Theoretical Computer Science*, vol. 84, no. 2, pp. 199–223, 1991.

41 A. Schrijver, *Combinatorial Optimization*. Springer, 2003.

## A Proofs

### A.1 Proof of Theorem 11

We describe a reduction from LTL satisfiability, which is PSPACE-hard [38]. As discussed in Remark 2, PSPACE-hardness applies already to LTL over finite computations.

Consider an LTL formula $\psi$. For simplicity, we assume that $\psi$ is over a single atomic proposition $p$. Indeed, LTL satisfiability is PSPACE-hard already for this fragment. We construct a flow network $N$ and an $\forall$LFL$_1$ formula $\varphi$ such that $N$ does not satisfy $\varphi$ iff some computation in $\{p, \neg p\}^*$ satisfies $\psi$. The network $N$ appears in Figure 3. Note that when $s$ has a flow of 1, then exactly one of $v_p$ and $v_{\neg p}$ has a flow of 1. Let $\psi'$ be the Flow-LTL formula obtained from $\psi$ be replacing all occurrences of $p$ by $\geq 1$. We define $\varphi = \mathcal{A}A(1 \to X\neg\psi')$. Thus, $\neg\varphi = \mathcal{E}E(1 \wedge X\psi')$.

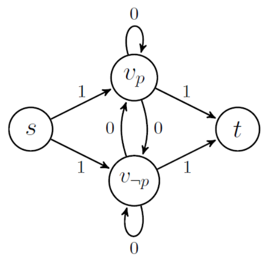


**Figure 3** The flow network $N$.

We prove that $N$ satisfies $\mathcal{E}E(1 \wedge X\psi')$ iff some computation in $\{p, \neg p\}^*$ satisfies $\psi$. Assume first that some computation $\pi$ in $\{p, \neg p\}^*$ satisfies $\psi$. Consider a flow $f$ in $N$ such that $f(v_p) = 1$ and $f(v_{\neg p}) = 0$, and consider the path in $N$ that encodes $\pi$ (starting from the second vertex in the path). For these flow and path, the formula $X\psi'$ holds, and therefore $N \models \mathcal{E}E(1 \wedge X\psi')$. For the other direction, assume that $N \models \mathcal{E}E(1 \wedge X\psi')$. Let $f$ and $\pi$ be the flow function and the path in $N$ that witness the satisfaction of $X\psi'$. The structure of $N$ guarantees that $f(v_p) + f(v_{\neg p}) = 1$. Hence, $\pi$ with $f$ encodes a computation in $\{p, \neg p\}^*$ that satisfies $\psi$, and we are done. Note that we can avoid using edges with capacities 0 by changing $\neg\varphi$ to $\mathcal{E}E(1 \wedge X0 \wedge XX\psi')$.

## A.2 Proof of the Reduction in Theorem 13

We prove that $\theta$ is satisfiable iff $N \models \mathcal{E}A\psi$.

Assume first that $\theta$ is satisfiable. Consider the flow function $f$ where for every $1 \leq i \leq n$, if $x_i$ holds in the satisfying assignment, then $f(x_i) = k$ and $f(\bar{x}_i) = 0$, and otherwise $f(x_i) = 0$ and $f(\bar{x}_i) = k$. For such $f$, all the paths in $N$ satisfy $\psi$.

Assume now that there is a flow function $f$ with which $N$ satisfies $A\psi$. For every $1 \leq i \leq n$, either we have $f(x_i) = k$ and $f(\bar{x}_i) = 0$ or we have $f(x_i) = 0$ and $f(\bar{x}_i) = k$. Consider the assignment $\tau$ to the variables $x_1, \ldots, x_n$ induced by $f$, where a literal $z \in Z$ holds in $\tau$ iff the flow in the corresponding vertex is positive. Since according to $\psi$ for every $i$ the flow in the vertex $C_i$ is positive, then every clause in $\theta$ contains at least one literal whose corresponding vertex has a positive flow. Hence, the assignment $\tau$ satisfies $\theta$.

## A.3 Proof of Lemma 15

We show that for every $\gamma > C_N$ we have $N \models \varphi[x_1 \leftarrow \gamma]$ iff $N \models \varphi[x_1 \leftarrow C_N]$. Every expression $g(X)$ in $\varphi$ is obtained from the variables in $X$ by applying the operators $+, *$, and possibly by using constants in $\mathbb{N}$. Hence, every such $g$ is monotonically increasing for every variable $x_i$. Let $\gamma_2, \ldots, \gamma_n$ be an assignment for the variables $x_2, \ldots, x_n$, respectively. Let $g_1(x_1) = g(x_1, \gamma_2, \ldots, \gamma_n)$. Let $\gamma_1$ be an integer bigger than $C_N$. Since $g_1$ is monotonically increasing, we have $g_1(C_N) \leq g_1(\gamma_1)$. If $g_1(C_N) < g_1(\gamma_1)$, then $g_1$ is not a constant function, and since it contains only the operators $+, *$ and constants in $\mathbb{N}$, we have $g_1(C_N) \geq C_N$. Therefore, if $g_1(C_N) \neq g_1(\gamma_1)$, then $g_1(\gamma_1) > g_1(C_N) \geq C_N$. Recall that for every flow function $f$ in $N$ and for every vertex $v \in V$, we have $f(v) < C_N$. Therefore, every state subformula $> g_1(x_1)$ and every state subformula $\geq g_1(x_1)$ holds for $\gamma_1$ iff it holds for $C_N$. Therefore, $N \models \varphi[x_1 \leftarrow \gamma]$ iff $N \models \varphi[x_1 \leftarrow C_N]$. Hence, we have $N \models \forall x_1 \varphi$ iff $N \models \varphi[x_1 \leftarrow \gamma]$ for every $0 \leq \gamma \leq C_N$.

## A.4 Proof of Lemma 17

Given $N$, we construct a new flow network $N'$ in which the attributions are on the edges. Thus, $N'$ is a flow network with lower bounds on the flow. We construct $N'$ by splitting every vertex $v \in V$ with attribution $[\gamma_l, \gamma_u]$ into two vertices in $N'$: $v_{in}$ and $v_{out}$. The vertices are connected by an edge $\langle v_{in}, v_{out} \rangle$ with attribution $[\gamma_l, \gamma_u]$. For every edge $\langle u, v \rangle \in E$, we add to $N'$ an edge $\langle u_{out}, v_{in} \rangle$ with attribution $[0, c(\langle u, v \rangle)]$. It is well known that deciding whether there exists a feasible flow and finding such a flow in a flow network with lower bounds can be done in polynomial time [9]. Also, the algorithm for finding such a feasible flow reduces the problem to a maximum-flow problem. Accordingly, if there is a feasible flow then the algorithm returns an integral one.