On the Path-Width of Integer Linear Programming

Constantin Enea[a], Peter Habermehl[a], Omar Inverso[b], Gennaro Parlato[c]

[a]*IRIF, University of Paris Diderot and CNRS, 75205 Paris 13, France*
[b]*Gran Sasso Science Institute, Viale Francesco Crispi 7, L'Aquila, Italy*
[c]*School of Electronics and Computer Science, University of Southampton, UK*

Abstract

We consider the feasibility problem of *integer linear programming* (ILP). We show that solutions of any ILP instance can be naturally represented by an FO-definable class of graphs. For each solution there may be many graphs representing it. However, one of these graphs is of path-width at most $2n$, where n is the number of variables in the instance. Since FO is decidable on graphs of bounded path-width, we obtain an alternative decidability result for ILP. The technique we use underlines a common principle to prove decidability which has previously been employed for automata with auxiliary storage. We also show how this new result links to automata theory and program verification.

Keywords: Integer linear programming, Bounded path-width, First-order logic on graphs, Automata

1. Introduction

Alur and Madhusudan in [1] have proposed nested words as a natural graph representation of runs of pushdown automata (PDA). A run is a sequence of moves which relate consecutive configurations of the PDA. A move is represented by a node, and nodes are linked through a linear order capturing the sequence of moves in the run. Further, nodes corresponding to matching push and pop moves are also linked together through (nested) matching edges. Thus, nested words naturally reflect the semantics of PDA.

Email addresses: cenea@liafa.univ-paris-diderot.fr (Constantin Enea), Peter.Habermehl@liafa.univ-paris-diderot.fr (Peter Habermehl), omar.inverso@gssi.infn.it (Omar Inverso), gennaro@ecs.soton.ac.uk (Gennaro Parlato)

This concept of representing runs with graphs has been extended to other classes of automata with multiple stacks and queues. For example, runs of multi-stack PDA can be represented as multiply-nested words, i.e. nested words with a nested relation for each stack. Similarly, runs of distributed automata can be represented with graphs. A distributed automaton consists of a finite number of PDAs communicating through unbounded queues. A natural graph representation for a run is composed of a finite number of nested words, each representing an execution of a single PDA, with additional edges modelling queues: a node representing the action of sending a message is linked to the corresponding node representing the action of receiving that message.

A surprising result by Madhusudan and Parlato shows that those graph representations straightforwardly lead to uniform decision procedures for several problems on these automata. In [2], it is shown that the emptiness problem for PDAs as well as several restrictions of multi-stack PDAs and distributed automata is decidable, as the class of graphs representing the runs of these automata has bounded tree-width, and furthermore it is definable in monadic second-order logic (MSO). Thus, checking the existence of an accepting run of those automata is equivalent to the satisfiability of the MSO formula characterizing runs on the class of graphs of bounded tree-width. The tree-width of a graph is a parameter that tells how close to a tree a graph is [3]. The problem of MSO satisfiability on graphs is undecidable in general, but decidable on the class of bounded tree-width graphs [4, 5].

Although this is a mathematical reduction from the emptiness problem for automata to MSO satisfiability on graphs, the novelty here is not the reduction itself. In fact, since the problem is decidable, one could first solve it and then write an MSO formula that is satisfiable on graphs of tree-width 1 if and only if the problem admits a positive answer. In contrast, the *principle* outlined in [2] is that a natural graph representation that logically captures the semantics of these automata–not containing algorithmic insights–is sufficient for decidability.

Among the problems that have been shown decidable using this principle we have: (1) state reachability problem [2], model-checking of LTL [6, 7], and *generalised* LTL [8] for various restrictions of multi-stack PDA [9, 10, 11, 12, 13, 14, 15], and (2) the reachability problem [2, 16] for subclasses of distributed automata that communicate through unbounded queues [17, 18]. The surprising aspect is that the new proofs are uniform and radically different from the ones previously proposed in the literature which are specifically crafted using different techniques on a case-by-case basis. This strengthens the intuition that a common principle governs the decidability of (those) problems. In general, the above principle could be lifted to decision problems. Although it may not be always applicable, it is in-

teresting to establish its generality or limits by looking at other decidability results known in the literature.

In this paper, we consider the *feasibility problem* for *integer linear programming* (ILP, for short) that asks whether, given a finite set I of linear constraints, there is an assignment of its variables such that all the constraints are satisfied[1]. We show that the decidability principle based on bounded tree-width graph representations applies to the ILP feasibility problem in a stronger sense as described below.

As a **first contribution** we give a *natural* graph representation for the solutions of an instance I of ILP. The nodes of the graph represent a unary encoding of the solution, i.e., each node is labelled with exactly one variable of I, and the number of nodes with the same label is the value of the corresponding variable in the solution. The edges are used to enforce the constraints of the system. For simplicity, consider a system with only one constraint, where each variable is associated with one coefficient. Depending on the sign of this coefficient each variable can contribute to the overall value of the constraint by either increasing or decreasing it. Each node will have a number of edges equal to the absolute value of the corresponding coefficient. We use edges to pair nodes whose corresponding coefficients have different signs. Thus, a graph with well-matched nodes is a solution. In case of multiple constraints, we reiterate the above mechanism for each constraint individually, labelling the edges with the constraint represented. Since multiple "matchings" are possible for the same solution, a solution may have several of those graphs representing it. We prove that the class of graphs representing the solutions of an instance I can be defined in first-order logic. See Figure 1 for an example of a solution for a two-constraints system.

In general, the class of graphs representing all solutions may have unbounded path-width. We show that, for any solution, there always exists a graph representing it of *path-width* at most $2n$, where n is the number of variables of I, and this constitutes the **second contribution** of the paper. The path-width of a graph measures its closeness to a path (rather than a tree, as for tree-width). This provides us with a restriction of the decidability principle outlined above for the case of ILP, where bounded path-width is already sufficient as opposed to the general case where the tree-width needs to be bounded.

As a **last contribution** we define, for each ILP instance I, a finite state au-

[1]W.l.o.g., we suppose that the variables are interpreted as positive integers and that I contains only equalities.

tomaton A_I over the alphabet of I's variables, such that the Parikh image [19] of A_I is exactly the set of all solutions of I. This construction relies on the proof of bounded path-width we provide. Furthermore, this automaton can also be seen as a Boolean program P_I of size linear in the size of I as opposed to the exponential size of A_I, such that I is feasible iff a given location in P_I is reachable. This gives a *symbolic* alternative to solve ILP using program verification tools.

Organization of the paper. In Section 2, we give basic definitions on graphs, tree-width, MSO on graphs, and the feasibility problem of ILP. In Section 3, we present the graph representation for ILP solutions, and give its FO characterisation. In Section 4, we present a new algorithm to solve the ILP feasibility problem. We exploit some of the properties of this algorithm to build a graph of bounded path-width for each solution of an ILP instance (in Section 5), and to define a finite-state automaton whose Parikh image is the set of all solutions of an ILP instance (in Section 6). We conclude with some remarks and future work in Section 7.

Related Work. Many approaches are known for solving the ILP feasibility problem, based on, e.g., branch-and-bound [20], the cutting-plane method [21], the LLL algorithm [22], the Omega test [23], finite-automata theory [24, 25, 26]. The latter defines finite-automata representations for the set of solutions of an ILP instance but, differently from our approach, they are based on representing the binary encodings of the integers involved in the solutions. The exponential bound on the minimal solutions of an ILP instance [27] implies that, for any feasible instance I, there is an exponential bound B, such that some (but not all) solutions have a graph representation of path-width bounded by B. We prove that there exists a bounded path-width graph representation for *each* solution of an instance I and the bound depends only on the number of variables of I.

This paper extends our previously published work [28].

2. Preliminaries

Given two integers i and j with $i \leq j$, we denote with $[i, j]$ the set of all integers k such that $i \leq k \leq j$.

Monadic second-order logic on graphs: Let Σ_V and Σ_E be two disjoint finite alphabets. A (Σ_V, Σ_E)-*labelled graph* is a structure

$$G = (V, E, \{V_a\}_{a \in \Sigma_V}, \{E_b\}_{b \in \Sigma_E}),$$

where V is a finite set of vertices, E is a finite multi-set of (undirected) edges represented by unordered pairs of elements of V, for each $a \in \Sigma_V$, $V_a \subseteq V$ is a set of a-labelled vertices, and, for each $b \in \Sigma_E$, $E_b \subseteq E$ is a multi-set of b-labelled edges. When $\Sigma_V = \Sigma_E = \emptyset$, G is called simply a graph. Let $v, v' \in V$, and $\pi = v_0, v_1, \ldots, v_t$ be any sequence of distinct vertices of G with $v = v_0$ and $v' = v_t$. A *path* in G from v to v' is any sequence π such that $\{v_{i-1}, v_i\} \in E$, for every $i \in [1, t]$. In the rest of the paper, we denote any edge of the form $\{u, v\}$ simply with a pair (u, v) with the meaning that it is an unordered pair.

We view graphs as logical structures, where V is the universe.

Each set of vertices V_a is a unary relation on vertices and each multi-set of edges E_b is a binary relation on vertices. *Monadic second-order logic* (*MSO* for short) is nowadays the standard logic to express properties on these structures. We fix a countable set of first-order variables (denoted by lower-case symbols, e.g., x, y) and a countable set of second-order variables (denoted by upper-case symbols, e.g. X, Y).

The first-order, resp., second-order, variables are interpreted as vertices, resp., sets of vertices, in the graph. An *MSO* formula φ is defined by the following grammar:

$$\varphi \quad \triangleq \quad x = y \mid V_a(x) \mid E_b(x, y) \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $a \in \Sigma_V$, $b \in \Sigma_E$, x, y are first-order variables, and X is a second-order variable. The semantics of *MSO* is defined as usual. First-order logic (*FO*, for short) is the restriction of *MSO* to formulas over first-order variables.

A class of (Σ_V, Σ_E)-labelled graphs C is *MSO-definable*, resp., *FO-definable*, if there is an *MSO*, resp., *FO*, formula φ such that C is exactly the class of (Σ_V, Σ_E)-labelled graphs that satisfy φ.

Tree/path-width of graphs: A *tree-decomposition* of a graph $G = (V, E)$ is a pair (T, bag), where $T = (N, \rightarrow)$ is a (finite) tree[2] and $bag : N \rightarrow 2^V$ is a function, that satisfies the following:

- For every $v \in V$, there is a vertex $n \in N$ such that $v \in bag(n)$.
- For every edge $(u, v) \in E$, there is a vertex $n \in N$ such that $u, v \in bag(n)$.

[2]A tree T is a graph having a special vertex called the *root* such that for every vertex v of T there is exactly one path from the root to v.

- If $u \in (bag(n) \cap bag(n'))$, for vertices $n, n' \in N$, then for every n'' that lies on the unique undirected path from n and n' in T, $u \in bag(n'')$.

A *path-decomposition* of a graph $G = (V, E)$ is a tree-decomposition (T, bag) such that T is a linear graph (i.e., a tree with exactly one leaf).

The *width* of a tree/path-decomposition of G is the size of the largest bag in it, minus one; i.e. $max_{n \in N}\{|bag(n)|\} - 1$. The *tree-width*, resp., *path-width*, of a graph is the *smallest* of the widths of any of its tree-decompositions, resp., path-decompositions. The notion of tree/path-width is well-defined since every finite graph admits a trivial tree/path-decomposition consisting of only one vertex labeled by the set of all vertices of the graph. The notions of tree/path-decomposition and tree/path-width are extended to (Σ_V, Σ_E)-labelled graphs by ignoring vertex and edge labels.

Satisfiability of *MSO*: The satisfiability problem for *MSO* is undecidable in general but it is decidable when restricting the class of models to graphs of bounded tree/path-width.

Theorem 1 (Seese [5])**.** *The problem of checking, given $k \in \mathbb{N}$ and $\varphi \in MSO$ over (Σ_V, Σ_E)-labelled graphs, whether there is a (Σ_V, Σ_E)-labelled graph G of tree-width at most k that satisfies φ, is decidable.*

Corollary 1. *Let C be an MSO definable class of (Σ_V, Σ_E)-labelled graphs. The problem of checking, given $k \in \mathbb{N}$ and an MSO-formula φ, whether there is a graph $G \in C$ of tree-width at most k that satisfies φ, is decidable.*

Integer Linear Programming (ILP): An *ILP instance* is a system of equations $A\vec{x} = \vec{b}$, where $A = (a_{j,i})_{j \in [1,m], i \in [1,n]}$ is a $m \times n$ matrix, $\vec{x} = (x_i)_{i \in [1,n]}$ is a vector of unknowns of size n, $\vec{b} = (b_j)_{j \in [1,m]}$ is a vector of size m, and all elements of A and $\vec{b}$ are integers [3]. The *ILP feasibility problem*, asks to check whether there exists an integer vector $\vec{s}$ of size n such that $A\vec{s} = \vec{b}$ ($\vec{s}$ is called a *solution* of $A\vec{x} = \vec{b}$). For the sake of simplicity, in this paper we only consider solutions composed of non-negative integers.

[3] We consider ILP instances in *standard form*. ILP instances expressed as inequalities, i.e., $A\vec{x} \leq \vec{b}$, can be converted to standard form by introducing slack variables.

Finite-state automata: A *deterministic finite-state automaton* is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta : Q \times \Sigma \mapsto Q$ is the (partial) transition function. Let Σ^* denote the set of all words over Σ. A word $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ iff there exists a sequence $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_{n+1}$ such that for each $i \in [0, n]$, $s_{i+1} \in \delta(s_i, a_i)$, $s_{n+1} \in F$, and $w = a_0 \cdot a_1 \cdot \ldots \cdot a_n$. The *language* of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of all words accepted by $\mathcal{A}$.

Parikh image: Let $\Sigma = \{a_1, a_2, \ldots, a_t\}$ be an alphabet and $L \subseteq \Sigma^*$. The *Parikh image* of L is a mapping $Parikh : L \mapsto \mathbb{N}^t$ that associates to each word $w \in L$ the tuple of natural numbers $(p_1, p_2, \ldots, p_t)$, where p_i is the number of occurrences of the symbol a_i in w, for every $i \in [1, t]$.

3. Graph representation for ILP solutions

Given an ILP instance $A\vec{x} = \vec{b}$, we define the set of graphs $\mathcal{G}[A\vec{x} = \vec{b}]$ having the property that each graph in $\mathcal{G}[A\vec{x} = \vec{b}]$ represents a solution of $A\vec{x} = \vec{b}$. On the other hand, for every solution of $A\vec{x} = \vec{b}$ there is at least one graph in $\mathcal{G}[A\vec{x} = \vec{b}]$ representing it (but possibly more than one). Furthermore, we show that $\mathcal{G}[A\vec{x} = \vec{b}]$ is FO definable, which gives a polynomial time reduction from the ILP feasibility problem to the satisfiability problem of FO.

We first give the intuition behind the graph representation of a solution, before we formalize and prove the results outlined above. Consider an ILP instance $A\vec{x} = \vec{b}$ with $\vec{x} = (x_1, x_2, \ldots, x_n)$. A graph G in $\mathcal{G}[A\vec{x} = \vec{b}]$, if any, has the following features. Each vertex of G is labelled with an index from the set $[0, n]$, and the tuple $\vec{s} = (s_1, s_2, \ldots, s_n)$ is a solution of $A\vec{x} = \vec{b}$, where s_i is the number of G vertices labelled with variable index i. Intuitively, all vertices of G labelled with i give a unary representation of s_i. Furthermore, G has a unique vertex labelled with 0, which represents the vector $\vec{b}$. To impose that $\vec{s}$ is a solution of $A\vec{x} = \vec{b}$, G is equipped with edges labelled with indices of constraints (each edge is labelled with a unique index). In order to satisfy the j-th constraint we impose that every vertex labelled with a variable index $i \in [1, n]$ is the end-point of $|a_{j,i}|$ edges labelled with j. Similarly, the unique vertex representing $\vec{b}$ is the end-point of $|b_j|$ edges labelled with j. A vertex also comes with a sign for each constraint: for an i-labelled vertex v and the j-th constraint (1) if $i \in [1, n]$ (it is a variable index) then v has the same sign as $a_{j,i}$, otherwise (2) v is the unique vertex labelled with 0, and has the opposite sign of b_j. All edges labelled with j concern the j-th constraint. Thus, we further impose that an edge labelled with j is always incident

to vertices with opposite signs. Intuitively, attaching to each vertex a set of end-points that connect to edges, the end-points of a vertex represent the constants of the matrix A in unary, and the arithmetic related to each constraint is done by just matching these end-points (through edges). In fact, for a constraint j each node labelled with $i \in [1, n]$ will contribute with $|a_{j,i}|$ edges with the same sign of $a_{j,i}$. A similar argument holds for the node labelled with 0. Therefore, imposing the matchings described above we make sure that $a_{j,1} \cdot x_1 + \cdots + a_{j,n} \cdot x_n = b_j$ holds. Since the matchings are imposed for all constraints we have that G faithfully represents a solution for all the linear constraints. It is worth noting that we do not deliberately impose how matchings are accomplished. Thus, the same solution $\vec{s}$ may have several graphs in $\mathcal{G}[A\vec{x} = \vec{b}]$ representing it. We now provide an example to illustrate this intuition.

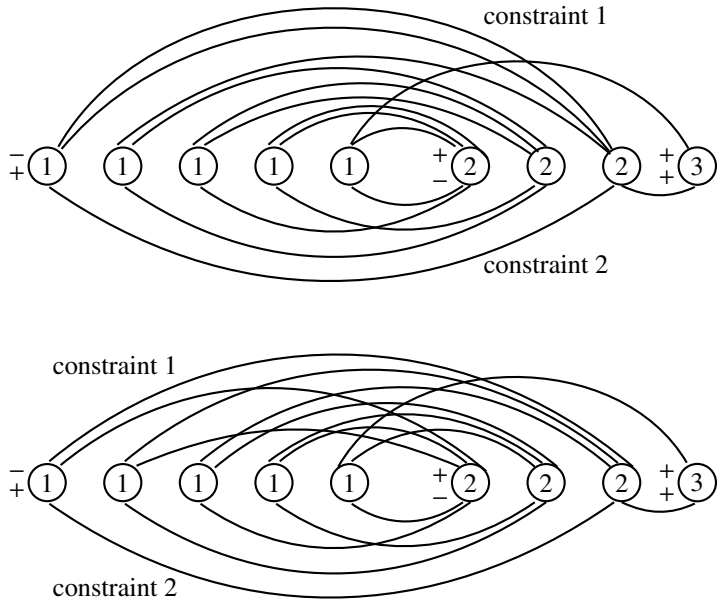

Figure 1: Two graph representations for the solution $x_1 = 5$, $x_2 = 3$, $x_3 = 1$ of $-2\,x_1 + 3\,x_2 + x_3 = 0$ and $x_1 - 2\,x_2 + x_3 = 0$. The edges above, resp., below, the vertices correspond to the first, respectively, the second, equation. The signs attached to the vertices are the signs of the corresponding coefficients in the two constraints. The vertex labelled by 0 is omitted because it has no incident edges.

Example 1. *The two graphs in Figure 1 represent the solution* $x_1 = 5$, $x_2 = 3$, $x_3 = 1$ *of the ILP instance* $-2\,x_1 + 3\,x_2 + x_3 = 0$ *and* $x_1 - 2\,x_2 + x_3 = 0$.

Definition 1. *Let* $A\vec{x} = \vec{b}$ *be an ILP instance.* $\mathcal{G}[A\vec{x} = \vec{b}]$ *is the set of all graphs* $G = (V, E, \{V_i\}_{i\in[0,n]}, \{E_j\}_{j\in[1,m]})$*, where:*

1. *V is a finite set of vertices and $\{V_i\}_{i\in[0,n]}$ defines a partition of V, i.e., for any $i \neq i' \in [0, n]$, $V_i \cap V_{i'} = \emptyset$ and $\bigcup_{i\in[0,n]} V_i = V$, and $|V_0| = 1$;*
2. *E is a finite multi-set of edges and $\{E_j\}_{j\in[1,m]}$ defines a partition of E, i.e., for any $j \neq j' \in [1, m]$, $E_j \cap E_{j'} = \emptyset$ and $\bigcup_{j\in[1,m]} E_j = E$;*
3. *if $(v, v') \in E_j$ with $v \in V_i$ and $v' \in V_{i'}$, then the signs of $a_{j,i}$ ($-b_j$ if $i = 0$) and $a_{j,i'}$ ($-b_j$ if $i = 0$) are different;*
4. *$|\{(v, v') \in E_j \mid v \in V_0\}| = |b_j|$ and for any $i \in [1, n]$ and $v \in V_i$, $|\{(v, v') \in E_j\}| = |a_{j,i}|$.*

Next, we show that every graph in $\mathcal{G}[A\vec{x} = \vec{b}]$ defines a solution of $A\vec{x} = \vec{b}$ and vice-versa. Let $sol : \mathcal{G}[A\vec{x} = \vec{b}] \to \mathbb{N}^n$ be a function that associates to every graph G in $\mathcal{G}[A\vec{x} = \vec{b}]$ the vector of natural numbers representing the number of vertices labelled with i, for each $i \in [1, n]$, i.e., $sol(G) = (|V_1|, \ldots, |V_n|)$.

Proposition 1. *The image of the function* $sol : \mathcal{G}[A\vec{x} = \vec{b}] \to \mathbb{N}^n$ *is exactly the set of all solutions of* $A\vec{x} = \vec{b}$.

Proof. Let $A\vec{x} = \vec{b}$ be an ILP instance and $G = (V, E, \{V_i\}_{i\in[0,n]}, \{E_j\}_{j\in[1,m]})$ be a graph in $\mathcal{G}[A\vec{x} = \vec{b}]$. We show that for every $j \in [1, m]$, $sol(G) = (|V_1|, \ldots, |V_n|)$ is a solution of the equation $a_{j,1} \cdot x_1 + \cdots + a_{j,n} \cdot x_n = b_j$. Let $a_{j,0} = -b_j$. The set of indices $[0, n]$ can be partitioned in two sets $\{p_1, \ldots, p_s\}$ and $\{n_1, \ldots, n_t\}$ s.t. for every $k \in [1, s]$, a_{j,p_k} is positive and for every $k \in [1, t]$, a_{j,n_k} is negative. By definition, all the edges of G labelled by j are between a vertex in $V_{p_1} \cup \ldots \cup V_{p_s}$ and a vertex in $V_{n_1} \cup \ldots \cup V_{n_t}$. Also, for every $i \in [0, n]$, the degree of every vertex in V_i equals $|a_{j,i}|$ and thus the number of edges labelled by j can be written as both

$$|V_{p_1}| \cdot a_{j,p_1} + \cdots + |V_{p_s}| \cdot a_{j,p_s} \text{ and } |V_{n_1}| \cdot |a_{j,n_1}| + \cdots + |V_{n_t}| \cdot |a_{j,n_t}|,$$

which proves that $sol(G)$ is a solution of $a_{j,1} \cdot x_1 + \cdots + a_{j,n} \cdot x_n = b_j$.

For the reverse, we show that for every solution $\vec{s} = (s_i)_{i\in[1,n]}$ of $A\vec{x} = \vec{b}$, there exists a graph $G = (V, E, \{V_i\}_{i\in[0,n]}, \{E_j\}_{j\in[1,m]})$ in $\mathcal{G}[A\vec{x} = \vec{b}]$ s.t. $sol(G) = \vec{s}$. Therefore, for every $i \in [1, n]$, the set V_i consists of s_i vertices. Then, for every

equation $a_{j,1} \cdot x_1 + \cdots + a_{j,n} \cdot x_n = b_j$ we consider the partition of $[0, n]$ into $\{p_1, \ldots, p_s\}$ and $\{n_1, \ldots, n_t\}$ exactly as above. We also consider that $a_{j,0} = -b_j$ and $s_0 = 1$. The fact that $\vec{s}$ is a solution implies that

$$s_{p_1} \cdot a_{j,p_1} + \cdots + s_{p_s} \cdot a_{j,p_s} = s_{n_1} \cdot |a_{j,n_1}| + \cdots + s_{n_t} \cdot |a_{j,n_t}|,$$

which shows that it is possible to define a multi-set of edges E_j satisfying the constraints in Definition 1. □

Proposition 1 implies that the feasibility of an ILP instance is reducible to the problem of checking the existence of a graph satisfying the properties in Definition 1.

Proposition 2. *An ILP instance $A\vec{x} = \vec{b}$ is feasible iff $\mathcal{G}[A\vec{x} = \vec{b}]$ is non-empty.*

Proof. An ILP instance $A\vec{x} = \vec{b}$ is feasible iff there exists a solution $\vec{s}$ such that $A\vec{s} = \vec{b}$, which by Proposition 1, is equivalent to the existence of a graph $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ such that $A \cdot sol(G) = \vec{b}$.

□

The following result shows that the class of graphs $\mathcal{G}[A\vec{x} = \vec{b}]$ from Definition 1 is definable in first-order logic.

Proposition 3. *For any ILP instance $A\vec{x} = \vec{b}$, there exists a first-order logic formula $\Phi[A\vec{x} = \vec{b}]$ such that for any graph G, $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ iff $G \models \Phi[A\vec{x} = \vec{b}]$.*

Proof. The formula $\Phi[A\vec{x} = \vec{b}]$ is defined as the conjunction of the formulae VertexLabels, EdgeLabels, Opposite, and Degree, which express the conditions (1), (2), (3), and (4) in Definition 1, respectively. These formulae are defined over the following set of predicates representing the vertex and edge labeling:

- V_i with $i \in [0, n]$ which holds for vertices labeled by i,
- $E_j(u, v)$ with $j \in [1, m]$ which holds for pairs of vertices (u, v) connected through an edge labeled by j,
- $E_j^k(u, v)$ with $j \in [1, m]$ and $k \in \mathbb{N}^*$, which holds iff there are exactly k edges labelled by j between u and v. For a given ILP instance, we will use only a bounded number of such predicates, where k is smaller than the maximum value in A or $\vec{b}$, in absolute value. This follows from the fact that the number of edges labeled by j incident to the same vertex can not be bigger than this maximum value.

The condition (1) in Definition 1 is expressed using the following formula:

$$\begin{array}{lcl}
\textsc{VertexLabels} & \triangleq & \textsc{VPartition}(V_0, \ldots, V_n) \wedge \textsc{Singleton}(V_0) \\
\textsc{VPartition}(V_0, \ldots, V_n) & \triangleq & \forall u.\ V_0(u) \oplus V_1(u) \oplus \ldots \oplus V_n(u) \\
\textsc{Singleton}(V_0) & \triangleq & \exists v. V_0(v) \wedge \forall w, w'.((V_0(w) \wedge V_0(w')) \rightarrow w = w'),
\end{array}$$

where $\oplus$ is the n-ary exclusive disjunction. $\textsc{VPartition}(V_0, \ldots, V_n)$ states that $V_0, \ldots, V_n$ form a partition of the set of vertices while $\textsc{Singleton}(V_0)$ states that V_0 is a singleton set.

The relation between the predicates $E_j(u, v)$ and $E_j^k(u, v)$, i.e. condition (2) in Definition 1, is expressed by the following formula:

$$\textsc{EdgeLabels} \triangleq \forall u, v. \bigwedge_{j \in [1,m]} E_j(u, v) \rightarrow (E_j^1(u, v) \oplus \ldots \oplus E_j^{max}(u, v))$$

where $\oplus$ is the max-ary exclusive disjunction. This formula states that, for any u and v connected through an edge labeled by j, there exists exactly one predicate $E_j^k(u, v)$ which holds.

For every $j \in [1, m]$, let pos_j be the set of i such that $a_{j,i} \geq 0$ together with 0, if $b_j \geq 0$. Analogously, let neg_j be the set of i such $a_{j,i} < 0$ together with 0, if $b_j < 0$. The formula $\textsc{Opposite}$ expresses condition (3) in Definition 1:

$$\begin{array}{lcl}
\textsc{Opposite} & \triangleq & \forall u, v. \bigwedge\limits_{j \in [1,m]} \Big(E_j(u, v) \rightarrow \textsc{opposite}_j(u, v)\Big) \\
\textsc{opposite}_j(u, v) & \triangleq & \left(\bigvee\limits_{i \in pos_j} V_i(u) \wedge \bigvee\limits_{i \in neg_j} V_i(v)\right) \vee \left(\bigvee\limits_{i \in neg_j} V_i(u) \wedge \bigvee\limits_{i \in pos_j} V_i(v)\right)
\end{array}$$

where $\textsc{opposite}_j(u, v)$ says that the coefficients of the variables x_i and $x_{i'}$ that label u and resp., v, in the jth constraint, have opposite signs.

Let max be the maximum value in A or $\vec{b}$, in absolute value. Then, $\textsc{Degree}$ expressing condition (4) in Definition 1 is defined by:

$$\textsc{Degree} \triangleq \forall u. \bigwedge_{\substack{i \in [0,n] \\ j \in [1,m]}} \textsc{degree}_{i,j}$$

$$\textsc{degree}_{i,j} \triangleq V_i(u) \rightarrow \bigvee_{\substack{z \leq |a_{j,i}| \\ t_1, \ldots, t_z > 0 \\ t_1 + \cdots + t_z = |a_{j,i}|}} \left(\begin{array}{c} \exists u_1, \ldots, u_z. distinct(u_1, \ldots, u_z) \\ \wedge E_j^{t_1}(u, u_1) \\ \wedge E_j^{t_2}(u, u_2) \\ \ldots \\ \wedge E_j^{t_z}(u, u_z) \end{array} \right)$$

where $\text{DEGREE}_{i,j}$ constrains the number of edges labeled by E_j incident to a vertex in V_i. Also, $\mathit{distinct}(u_1, \ldots, u_z)$ is the conjunction of all $u_i \neq u_{i'}$ with $1 \leq i \neq i' \leq z$. □

Example 2. *Consider the ILP instance from Example 1:*

$$\begin{array}{rrrr} -2x_1 & +3x_2 & +x_3 & = 0 \\ x_1 & -2x_2 & +x_3 & = 0 \end{array}$$

The first-order formula describing the graph representations of this ILP instance is the conjunction of the following sub-formulae:

$$\begin{aligned}
\text{VPARTITION}(V_0, V_1, V_2, V_3) &\triangleq \forall u.\ V_0(u) \oplus V_1(u) \oplus V_2(u) \oplus V_3(u) \\
\text{SINGLETON}(V_0) &\triangleq \exists v. V_0(v) \wedge \forall w, w'.((V_0(w) \wedge V_0(w')) \rightarrow w = w') \\
\text{EDGELABELS} &\triangleq \forall u, v. \bigwedge_{j \in [1,2]} E_j(u,v) \rightarrow (E_j^1(u,v) \oplus E_j^2(u,v) \oplus E_j^3(u,v)) \\
\text{OPPOSITE} &\triangleq \forall u, v.\ E_1(u,v) \rightarrow (((V_2(u) \vee V_3(u) \vee V_0(u)) \wedge V_1(v)) \\
&\qquad \vee\ (V_1(u) \wedge (V_2(v) \vee V_3(v) \vee V_0(v)))) \\
&\quad \wedge\ \forall u, v.\ E_2(u,v) \rightarrow (((V_1(u) \vee V_3(u) \vee V_0(u)) \wedge V_2(v)) \\
&\qquad \vee\ (V_2(u) \wedge (V_1(v) \vee V_3(v) \vee V_0(v)))) \\
\text{DEGREE} &\triangleq \forall u. \bigwedge_{\substack{i \in [0,3] \\ j \in [1,2]}} \text{DEGREE}_{i,j}
\end{aligned}$$

For conciseness, we give just one instance of the sub-formulae $\text{DEGREE}_{i,j}$*. For example,* $\text{DEGREE}_{1,1}$ *is defined by:*

$$\begin{aligned}
V_1(u) \rightarrow\ & \exists u_1.\ E_1^2(u, u_1) \\
& \vee\ \exists u_1, u_2.\ \mathit{distinct}(u_1, u_2) \wedge E_1^1(u, u_1) \wedge E_1^1(u, u_2).
\end{aligned}$$

4. Algorithm

In this section we present a new algorithm, BALANCE, to solve the ILP feasibility problem. We exploit some of the properties of this algorithm to build a graph of bounded path-width for each solution of an ILP instance (Section 5), and to define a finite-state automaton whose Parikh image is the set of all solutions of an ILP instance (Section 6).

In the rest of the section, I denotes a system of equations $A\vec{x} = \vec{b}$ where $\vec{x} = (x_i)_{i\in[1,n]}$, $\vec{s}$ is a solution of I, and $\vec{c_i}$ is the i-th column of the matrix A. Without loss of generality, we assume that $\vec{b}$ is a tuple of non-negative integers.

Algorithm Balance takes as input the matrix A and the vector $\vec{b}$ and returns a solution of I, if any. To give the intuition behind this algorithm, we start by explaining a simplistic version called ILP-Alg shown in Figure 2(a). ILP-Alg uses a vector $\vec{r} = (r_1, \ldots, r_m)$ of counter variables, where r_i is a counter associated with the i-th equation of I. It also uses a vector of variables $\vec{s} = (s_1, \ldots, s_n)$ to store a solution. All variables are initially set to zero. The algorithm is iterative. At each iteration, it nondeterministically selects a subset of A columns and adds them to the vector of counters $\vec{r}$. Furthermore, s_i is incremented if the i-th column is selected. When the valuation of the counters in $\vec{r}$ is $\vec{b}$, the algorithm is allowed to exit the loop and terminate by returning the valuation of $\vec{s}$, which is a solution. Observe that the value of s_i corresponds to the number of times column i has been added to $\vec{r}$.

Algorithm ILP-Alg loops forever when the instance does not admit any solution. On the other hand, when a solution exists, there is always an execution of the algorithm that returns it. Therefore, the algorithm can be used as a semi-decision procedure.

We now show that ILP-Alg can be turned into a decision procedure for the ILP feasibility problem, which is the algorithm Balance announced above. We illustrate that if a solution exists for I, we can adequately select a subset of columns at each iteration in a way that the values of the counters will range over finite intervals whose endpoints are only determined by the constants in A and $\vec{b}$. In particular, counter r_j will range within the interval $[l_j, u_j]$, where l_j is the sum of all negative elements in the j-th row of A and u_j is the sum of all positive elements in the j-row of A, plus b_j. This condition guarantees that the algorithm can be used as a decision procedure by restricting its state space, which is now finite. The algorithm Balance is shown in Figure 2(b). Lines 5-8 set the bounds of the interval associated with each counter. Line 16 blocks the execution of the algorithm if any counter in $\vec{r}$ exceeds its bounds.

We now prove the correctness of Balance by showing that for any solution $\vec{s} = (s_1, \ldots, s_n)$ of I there always exists a way to select the subsets of columns at each iteration of ILP-Alg such that the counters $\vec{r}$ do not exceed the bounds, and ILP-Alg is able to return $\vec{s}$. For an execution π of ILP-Alg, let $\pi_C = S_1, \ldots, S_t$ be the sequence of sets of column indices selected at each iteration along π. Note that for any sequence $S_1, \ldots, S_t$ of sets of column indices, there is at most one

```
ILP-ALG(A, b⃗)
1   r⃗ = (r_1, ..., r_m) ← 0⃗
2   s⃗ = (s_1, ..., s_n) ← 0⃗
3
4
5
6
7
8
9   while ((r⃗ ≠ b⃗) ∨ NONDET()) do
10      for i ← 1 to n do
11          if NONDET() then
12              r⃗ ← r⃗ + c⃗_i
13              s_i ← s_i + 1
14          end
15      end
16
17  end
18
19  return s⃗
```

(a)

```
BALANCE(A, b⃗)
    r⃗ = (r_1, ..., r_m) ← 0⃗
    s⃗ = (s_1, ..., s_n) ← 0⃗
    l⃗ = (l_1, ..., l_m)
    u⃗ = (u_1, ..., u_m)
    for j ← 1 to m do
        l_j ← Σ_{i∈[1,n], a_{j,i}<0} (a_{j,i})
        u_j ← Σ_{i∈[1,n], a_{j,i}>0} (a_{j,i}) + b_j
    end
    while ((r⃗ ≠ b⃗) ∨ NONDET()) do
        for i ← 1 to n do
            if NONDET() then
                r⃗ ← r⃗ + c⃗_i
                s_i ← s_i + 1
            end
        end
        assume( r⃗ ≥ l⃗ ∧ r⃗ ≤ u⃗ )
    end

    return s⃗
```

(b)

Figure 2: Algoritms ILP-ALG and BALANCE.

execution π of ILP-Alg such that $\pi_C = S_1, \ldots, S_t$.

We define a particular execution π where ILP-Alg returns $\vec{s}$, and which is also an execution of Balance, called $\vec{s}$-execution, through its sequence π_C. This sequence $S_1, \ldots, S_{s_l}$ with $s_l = \max\{s_1, \ldots, s_n\}$, denoted $seq_{\vec{s}}$, is defined as follows. Intuitively, the idea is to *evenly distribute* s_i across the sequence of sets, for any $i \in [1, n]$. More precisely, for $i \in [1, n]$ and $k \in [1, s_l]$, define

$$distribute_{\vec{s}}(i, k) = \left\lceil k \cdot \frac{s_i}{s_l} \right\rceil \tag{1}$$

representing the number of sets in $S_1, \ldots, S_k$ containing index i. Define $seq_{\vec{s}}$ as follows. S_1 is the set of all indices i such that $distribute_{\vec{s}}(i, 1) = 1$. For $k \in [2, s_l]$, S_k is the set of all indices $i \in [1, n]$ such that $distribute_{\vec{s}}(i, k) - distribute_{\vec{s}}(i, k-1) = 1$.

Example 3. *For the solution $\vec{s} = (5, 3, 1)$ of the ILP instance of Example 1, the sequence $seq_{\vec{s}}$, and the sequence of counter values (r_1, r_2) at the end of each iteration along the $\vec{s}$-execution is as follows:*

$$\begin{array}{ccccc} (0,0) & \rightarrow_{S_1=\{1,2,3\}} & (2,0) & & \textbf{[iteration 1]} \\ (2,0) & \rightarrow_{S_2=\{1,2\}} & (3,-1) & & \textbf{[iteration 2]} \\ (3,-1) & \rightarrow_{S_3=\{1\}} & (1,0) & & \textbf{[iteration 3]} \\ (1,0) & \rightarrow_{S_4=\{1,2\}} & (2,-1) & & \textbf{[iteration 4]} \\ (2,-1) & \rightarrow_{S_5=\{1\}} & (0,0) & & \textbf{[iteration 5]} \end{array}$$

Notice that counters range over the intervals defined above, where $r_1 \in [-2, +4]$, and $r_2 \in [-2, +2]$.

We are now ready to prove the boundedness of the counters $\vec{r}$ along the $\vec{s}$-execution of ILP-Alg, which implies that it is also an execution of Balance.

Lemma 1. *Let $A\vec{x} = \vec{b}$ be a system of linear equations with variables $\vec{x} = (x_i)_{i \in [1,n]}$, $\vec{s}$ be a solution of the system, and r_j^k be the valuation of variable r_j at the end of the k-th iteration of the $\vec{s}$-execution of* ILP-Alg. *Then,*

$$\sum_{\substack{i \in [1,n], \\ a_{j,i} < 0}} (a_{j,i}) \;<\; r_j^k \;<\; b_j + \sum_{\substack{i \in [1,n], \\ a_{j,i} > 0}} (a_{j,i})$$

Proof. Observe that the definition of $distribute_{\vec{s}}(i, k)$ in (1) can be reformulated as

$$k \cdot \frac{s_i}{s_l} + z_{i,k} \qquad \text{for some real } z_{i,k} \in [0, 1[$$

therefore:

$$\begin{aligned} r_j^k &= \sum_{i=1}^{n} \left(distribute_{\vec{s}}(i,k) \cdot a_{j,i} \right) \\ &= \sum_{i=1}^{n} \left(\left(k \cdot \frac{s_i}{s_l} + z_{i,k} \right) \cdot a_{j,i} \right) \\ &= \sum_{i=1}^{n} \left(k \cdot \frac{s_i}{s_l} \cdot a_{j,i} \right) + \sum_{i=1}^{n} \left(z_{i,k} \cdot a_{j,i} \right) \\ &= \frac{k}{s_l} \cdot \sum_{i=1}^{n} \left(s_i \cdot a_{j,i} \right) + \sum_{i=1}^{n} \left(z_{i,k} \cdot a_{j,i} \right) \\ &= \frac{k}{s_l} \cdot b_j + \sum_{i=1}^{n} \left(z_{i,k} \cdot a_{j,i} \right) \end{aligned}$$

Note that $\frac{k}{s_l} \leq 1$ (as $k \in [1, s_l]$), $b_j \geq 0$, and $z_{i,k} \in [0, 1[$, thereby from the relation above it is easy to see that that r_j^k will range within the bounds defined above. □

Since $distribute_{\vec{s}}(i, s_l) = s_i$ for any $i \in [1, n]$, i.e., the i-th component of the solution $\vec{s}$, it is guaranteed that the valuation of $\vec{r}$ after s_l iterations matches $\vec{b}$ and therefore a solution can be returned by the algorithm in Figure 2. This, along with Lemma 1 gives the main result of this section.

Theorem 2. *Algorithm* Balance *is a nondeterministic finite-state algorithm that solves the ILP feasibility problem.*

4.1. On $\vec{s}$-executions of Balance

In the rest of this section we give some properties on $\vec{s}$-executions of Balance, which are used in Section 5.

Let $r_j^1, \ldots, r_j^{s_l - 1}$ be the values of the counter variable r_j after each iteration (see Lemma 1 for a formal definition). Intuitively, the counters r_j^k indicate "how far" (in the j-th constraint) the solution is after k iterations.

From the proof of Lemma 1, for all $k \in [1, s_l - 1]$,

$$r_j^k = \frac{k}{s_l} \cdot b_j + \sum_{i=1}^{n} (z_{i,k} \cdot a_{j,i}), \qquad \text{with } z_{i,k} \in [0, 1[.$$

To unify notation, define $z_{0,k} = \frac{k}{s_l} (< 1)$ and $a_{j,0} = b_j$. Thus,

$$r_j^k = \sum_{i=0}^{n} (z_{i,k} \cdot a_{j,i}), \qquad \text{with } z_{i,k} \in [0, 1[.$$

We are interested in the values $(z_{i,k} \cdot a_{j,i})$ for $i \in [0, n]$, $j \in [1, m]$, $k \in [1, s_l - 1]$ which correspond to the "contribution" to r_j^k of each coefficient. Although the r_j^k are always integers, each of these values might not be an integer. In Section 5 we need integer approximations (they will correspond to a certain number of edges in solution graphs) of these values while keeping their sum r_j^k the same. To obtain these integer approximations, let us first define

$$pos_j = \{i \in [0, n] \mid a_{j,i} \geq 0\} \quad \text{and} \quad neg_j = \{i \in [1, n] \mid a_{j,i} < 0\}.$$

For each $k \in [1, s_l - 1]$, define for each value r_j^k its positive part

$$r_{j,pos}^k = \sum_{p \in pos_j} (z_{p,k} \cdot a_{j,p})$$

and its negative part

$$r_{j,neg}^k = \sum_{p \in neg_j} (z_{p,k} \cdot a_{j,p}).$$

Obviously, $r_j^k = r_{j,pos}^k + r_{j,neg}^k$.

In the example, we have the following successive values for

$$(r_{1,pos}^k, r_{2,pos}^k) \ : \left(2, \frac{4}{5}\right), \left(3, \frac{3}{5}\right), \left(1, \frac{2}{5}\right), \left(2, \frac{1}{5}\right)$$

and for

$$(r_{1,neg}^k, r_{2,neg}^k) \ : \left(0, -\frac{4}{5}\right), \left(0, -\frac{8}{5}\right), \left(0, -\frac{2}{5}\right), \left(0, -\frac{6}{5}\right).$$

We now prove the following Lemma which plays an important role in Section 5 and provides integer approximations for the values $(z_{i,k} \cdot a_{j,i})$ along with a further condition.

Lemma 2. *For all $i \in [0, n]$, $j \in [1, m]$ and $k \in [1, s_l - 1]$, there exist $c_{j,i}^k \in \left\{\lfloor z_{i,k} \cdot a_{j,i} \rfloor, \lceil z_{i,k} \cdot a_{j,i} \rceil\right\}$ such that*

(a) $\sum_{p \in pos_j} c_{j,p}^k = \left\lceil r_{j,pos}^k \right\rceil$,

(b) $\sum_{p \in neg_j} c^k_{j,p} = \left\lfloor r^k_{j,neg} \right\rfloor$, *and*

(c) *if* $k < s_l - 1$ *and* $distribute_{\vec{s}}(i,k) = distribute_{\vec{s}}(i,k+1)$, *then* $|c^k_{j,i}| \geq |c^{k+1}_{j,i}|$.

Furthermore, $\sum_{i=0}^{n} c^k_{j,i} = r^k_j$ *and* $|c^k_{j,i}| \leq |a_{j,i}|$.

Proof. The proof does not depend on j; assume that j is fixed. We only show how to choose the $c^k_{j,i}$'s for $i \in pos_j$. The case $i \in neg_j$ is symmetric. Notice that clearly,

$$\sum_{p \in pos_j} \lfloor z_{p,k} \cdot a_{j,p} \rfloor \leq \lceil r^k_{j,pos} \rceil \leq \sum_{p \in pos_j} \lceil z_{p,k} \cdot a_{j,p} \rceil.$$

We first set $c^k_{j,i} = \lceil z_{i,k} \cdot a_{j,i} \rceil$, for all $k \in [1, s_l - 1]$. This guarantees $\sum_{p \in pos_j} c^k_{j,p} \geq \lceil r^k_{j,pos} \rceil$, for all $k \in [1, s_l - 1]$. Then, we have $\sum_{p \in pos_j} \lceil z_{p,k} \cdot a_{j,p} \rceil = \sum_{p \in pos_j} c^k_{j,p} = f_k + \lceil r^k_{j,pos} \rceil$ for some natural number f_k. We have to change f_k many of the $c^k_{j,i}$'s to $\lfloor z_{i,k} \cdot a_{j,i} \rfloor$ (provided that $z_{i,k} \cdot a_{j,i} \notin \mathbb{N}$), i.e., we have to choose f_k many i's from pos_j. We can not just arbitrarily choose any of them at each step, as condition (c) might be violated.

We now define for each $k \in [1, s_l - 1]$ sets $F_k \subseteq pos_j$ of i's for which $c^k_{j,i}$ is set to $\lfloor z_{i,k} \cdot a_{j,i} \rfloor$ and then we show that conditions (a) and (c) are satisfied. To do that we define first for each $k \in [2, s_l - 1]$ the set of indices $X_k = \{i \in pos_j \mid distribute_{\vec{s}}(i, k-1) = distribute_{\vec{s}}(i,k) \wedge \lfloor z_{i,k-1} \cdot a_{j,i} \rfloor = \lfloor z_{i,k} \cdot a_{j,i} \rfloor\}$. Intuitively, indices in X_k which appear in F_{k-1} must also appear in F_k to satisfy condition (c). Due to the definition of *distribute* (see (1)) we have for all $i \in X_k$

$$z_{i,k-1} \cdot a_{j,i} \quad > \quad z_{i,k} \cdot a_{j,i} \tag{2}$$

Then, we define for each $i \in pos_j$ and $k \in [1, s_l - 2]$, $m_{i,k}$ to be the smallest $k' > k$ such that $i \notin X_{k'}$. Then, we choose F_1 of size f_1 to be a set of i's (which is not necessarily unique) in such a way that the $m_{i,1}$'s are minimal (i.e. it is not possible to replace an i in F_1 by an i' such that $m_{i',1} < m_{i,1}$). Intuitively, this ensures that the set of indices out of X_2 (resp. X_3 and so on) which must also be in F_2 (resp. F_3 and so on) does not get too big. Inductively for $k \in [2, s_l - 1]$, F_k is now defined as the disjoint union of G_k and H_k where $G_k = F_{k-1} \cap X_k$ and H_k is chosen to be a set of i's such that $|H_k| = f_k - |G_k|$ and the corresponding $m_{i,k}$'s are minimal.

Clearly, condition (c) is satisfied, since F_k contains G_k. However, it remains to show that $|G_k| \leq f_k$ for all $2 \leq k < s_l$ which implies that H_k and hence F_k are

well-defined. It is enough to show

$$\sum_{p\in G_k} \left\lfloor z_{p,k} \cdot a_{j,p} \right\rfloor + \sum_{p\in pos_j \setminus G_k} \left\lceil z_{p,k} \cdot a_{j,p} \right\rceil \;\geq\; \left\lceil r^k_{j,pos} \right\rceil \tag{3}$$

since that implies $\lceil r^k_{j,pos} \rceil \leq \sum_{p\in G_k} \lfloor z_{p,k} \cdot a_{j,p} \rfloor + \sum_{p\in pos_j \setminus G_k} \lceil z_{p,k} \cdot a_{j,p} \rceil = -|G_k| + \sum_{p\in G_k} \lceil z_{p,k} \cdot a_{j,p} \rceil + \sum_{p\in pos_j \setminus G_k} \lceil z_{p,k} \cdot a_{j,p} \rceil = -|G_k| + f_k + \lceil r^k_{j,pos} \rceil$ and therefore $|G_k| \leq f_k$.

We show (3) by contradiction. Suppose (3) does not hold. Then, $G_k \neq \emptyset$ and we take the smallest k such that $\sum_{p\in G_k} \lfloor z_{p,k} \cdot a_{j,p} \rfloor + \sum_{p\in pos_j \setminus G_k} \lceil z_{p,k} \cdot a_{j,p} \rceil < \lceil r^k_{j,pos} \rceil$. We have $\lceil r^k_{j,pos} \rceil = \left\lceil \sum_{p\in pos_j} (z_{p,k} \cdot a_{j,p}) \right\rceil$. Then,

$$\left\lceil \sum_{p\in pos_j} (z_{p,k} \cdot a_{j,p}) \right\rceil > \sum_{p\in G_k} \lfloor z_{p,k} \cdot a_{j,p} \rfloor + \sum_{p\in X_k \setminus G_k} \lceil z_{p,k} \cdot a_{j,p} \rceil + \sum_{p\in pos_j \setminus X_k} \lceil z_{p,k} \cdot a_{j,p} \rceil.$$

This implies

$$\left\lceil \sum_{p\in X_k} (z_{p,k} \cdot a_{j,p}) \right\rceil \;>\; \sum_{p\in G_k} \lfloor z_{p,k} \cdot a_{j,p} \rfloor + \sum_{p\in X_k \setminus G_k} \lceil z_{p,k} \cdot a_{j,p} \rceil.$$

We have

$$\sum_{p\in G_k} \lfloor z_{p,k} \cdot a_{j,p} \rfloor + \sum_{p\in X_k \setminus G_k} \lceil z_{p,k} \cdot a_{j,p} \rceil \;=\; \sum_{p\in G_k} \lfloor z_{p,k-1} \cdot a_{j,p} \rfloor + \sum_{p\in X_k \setminus G_k} \lceil z_{p,k-1} \cdot a_{j,p} \rceil$$

which entails

$$\left\lceil \sum_{p\in X_k} (z_{p,k} \cdot a_{j,p}) \right\rceil \;>\; \sum_{p\in G_k} \lfloor z_{p,k-1} \cdot a_{j,p} \rfloor + \sum_{p\in X_k \setminus G_k} \lceil z_{p,k-1} \cdot a_{j,p} \rceil. \tag{4}$$

Furthermore, due to our way of defining F_{k-1} we have

$$pos_j \setminus X_k \subseteq F_{k-1} \tag{5}$$

Suppose to the contrary that there is an $i \in pos_j$, $i \notin X_k$ and $i \notin F_{k-1}$. Then, we have $m_{i,k-1} = k$ because $i \notin X_k$ and for all i' in $G_k = (F_{k-1} \cap X_k) \neq \emptyset$ we have $m_{i',k-1} \geq k+1$. Now,

- if $k = 2$, then we have a contradiction with the definition of F_1 as i should be in F_1 instead of one of the i' since $m_{i,1} < m_{i',1}$.

- if $k > 2$ and there exists an $i' \in H_{k-1}$, then we have a contradiction with the definition of H_{k-1} as i should be in H_{k-1} instead of i' since $m_{i,k-1} < m_{i',k-1}$.
- if $k > 2$ and $H_{k-1} = \emptyset$, then we have $F_{k-1} = F_{k-2} \cap X_{k-1}$. Therefore $i \notin X_{k-1}$ and $i \notin F_{k-2}$ and we continue inductively until we reach a contradiction.

With (5) and $G_k = F_{k-1} \cap X_k$ we have

$$\left\lceil r^{k-1}_{j,pos} \right\rceil = \left\lceil \sum_{p \in pos_j} (z_{p,k-1} \cdot a_{j,p}) \right\rceil = \sum_{p \in F_{k-1}} \lfloor (z_{p,k-1} \cdot a_{j,p}) \rfloor + \sum_{p \in X_k \setminus G_k} \lceil (z_{p,k-1} \cdot a_{j,p}) \rceil$$

which implies

$$\left\lceil \sum_{p \in X_k} (z_{p,k-1} \cdot a_{j,p}) \right\rceil \leq \sum_{p \in G_k} \lfloor (z_{p,k-1} \cdot a_{j,p}) \rfloor + \sum_{p \in X_k \setminus G_k} \lceil z_{p,k-1} \cdot a_{j,p} \rceil$$

which together with (2) entailing $\left\lceil \sum_{p \in X_k} (z_{p,k-1} \cdot a_{j,p}) \right\rceil \geq \left\lceil \sum_{p \in X_k} (z_{p,k} \cdot a_{j,p}) \right\rceil$ and with (4) is a contradiction and therefore (3) is true.

Finally, (a) and (b) together guarantee $\sum_{i=0}^{n} c^k_{j,i} = r^k_j$ and we have $|c^k_{j,i}| \leq |a_{j,i}|$, as $z_{i,k} < 1$. □

In the example we choose as successive values for

$$(c^k_{1,0}, c^k_{1,1}, c^k_{1,2}, c^k_{1,3}) \ : (0,0,2,0), (0,0,3,0), (0,0,1,0), (0,0,2,0)$$

and we choose as successive values for

$$(c^k_{2,0}, c^k_{2,1}, c^k_{2,2}, c^k_{2,3}) \ : (0,0,-1,1), (0,0,-2,1), (0,0,-1,1), (0,0,-2,1).$$

Notice that the first component is always 0 since $b_1 = b_2 = 0$, and the second component is always 0 since $s_1 = s_l$ and therefore $z_{1,k} = 0$ for all $k \in [1, s_l - 1]$.

5. Bounded Path-width

For each solution $\vec{s}$ of a system of linear equations $A\vec{x} = \vec{b}$ with n variables, $\mathcal{G}[A\vec{x} = \vec{b}]$ may contain several graphs G with $\vec{s} = sol(G)$. All of those have the same number of vertices with the same label as well as the same number of edges with the same label. In this section, we show (using the results from Section 4) that among all graphs in $\mathcal{G}[A\vec{x} = \vec{b}]$ representing $\vec{s}$, there is one whose path-width is bounded by $2 \cdot n$. Thus, the decidability of the ILP feasibility problem can be directly derived from the decidability of FO on the class of bounded path-width graphs.

Lemma 3. *Let $A\vec{x} = \vec{b}$ be a system of linear equations with n variables. For each solution $\vec{s}$ of $A\vec{x} = \vec{b}$, there is a graph $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ with $sol(G) = \vec{s}$ and whose path-width is at most $2 \cdot n$.*

The proof of this lemma is given later in the section.

From Lemma 3 and Proposition 2, we obtain the main result of this section.

Theorem 3. *An ILP instance $A\vec{x} = \vec{b}$ with n variables is feasible if and only if there exists a graph $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ of path-width bounded by $2 \cdot n$.*

Corollary 2. *An ILP instance $A\vec{x} = \vec{b}$ with n variables is feasible if and only if the first order formula $\Phi[A\vec{x} = \vec{b}]$ is satisfiable on the class of graphs of path-width bounded by $2 \cdot n$.*

In the rest of the section we give a proof of Lemma 3, which is organized as follows. Out of the many graphs $\mathcal{G}[A\vec{x} = \vec{b}]$ we will construct one graph $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ with path-width bounded by $2 \cdot n$ by first defining in Section 5.1 a partition $\mathcal{P}$ of vertices of G. This partition is derived from the $\vec{s}$-execution of Balance of Section 4 and it induces a path-decomposition whose width depends on the edges. Then, starting from the partition $\mathcal{P}$ we show in Section 5.2 how to define the set of edges of G such that the induced path-decomposition has width bounded by $2 \cdot n$.

From now on, we assume that $A\vec{x} = \vec{b}$ is a system of linear equations, and $\vec{s} = (s_1, \ldots, s_n)$ is a solution of $A\vec{x} = \vec{b}$.

5.1. Path decomposition based on vertex partition

We define a path decomposition of any graph in $\mathcal{G}[A\vec{x} = \vec{b}]$ based on the notion of vertex partitions in *special form*.

Definition 2. *Let $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ with $G = (V, E, \{V_i\}_{i\in[0,n]}, \{E_j\}_{j\in[1,m]})$, and let $\mathcal{P} = \{V^k\}_{k\in[1,t]}$ be a partition of V, for some $t \in \mathbb{N}$. $\mathcal{P}$ is a* vertex partition in special form *of G if $|V^k \cap V_i| \leq 1$, for any $k \in [1, t]$ and $i \in [0, n]$, i.e., no two vertices of V^k are labeled with the same variable index. Furthermore, the unique vertex labeled by 0 is in V^1.*

Let $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ and $\mathcal{P} = \{V^k\}_{k\in[1,t]}$ be a vertex partition in special form of G. We now define a path decomposition (L, bag) of G built from $\mathcal{P}$ as follows. For $V' \subseteq V$, a vertex $v \in V'$ is *fully matched* in the subgraph of G induced by V', denoted $G(V')$, if $G(V')$ contains all the edges of G incident to v. The path decomposition (L, bag) is given by the line graph L formed by the sequence of vertices

$l_1, \ldots, l_t$, where $bag(l_k)$ is defined as follows. For the sake of simplicity, denote $bag(l_k)$ as $B^k_{\mathcal{P}}$, for any $k \in [1, t]$. We define the sequence of bags $B^1_{\mathcal{P}}, B^2_{\mathcal{P}}, \ldots, B^t_{\mathcal{P}}$ recursively:

- $B^1_{\mathcal{P}} = V^1$;
- for $k \in [2, t]$,

$$B^k_{\mathcal{P}} = \left\{ v \in B^{k-1}_{\mathcal{P}} \mid v \text{ is not fully matched in } G(V^1 \cup \cdots \cup V^{k-1}) \right\} \cup V^k.$$

It starts with V^1, and in subsequent steps, say at step k with $k > 1$, $B^k_{\mathcal{P}}$ is the set obtained from the previous bag, i.e., $B^{k-1}_{\mathcal{P}}$, by eliminating all fully matched vertices of $G(V^1 \cup \cdots \cup V^{k-1})$, and then adding the vertices of V^k. Therefore, (L, bag) is a path decomposition of G:

1. each vertex of G is in at least one bag as $\mathcal{P}$ is a partition of the G vertices,
2. once a vertex is in a bag, it will be removed only when all its incident edges in G are contained in the subgraph induced by $V^1 \cup \cdots \cup V^{k-1}$, guaranteeing that for every edge $(u, v) \in E$ there is a bag containing its endpoints, and
3. once a vertex is removed from a bag it will never be inserted in any subsequent bag.

Since at most one vertex with label 0 is contained in a solution graph, it entails that $|V^1| \leq n + 1$ and $|V^k| \leq n$ for every $k \in [2, t]$. Thus, the following property holds.

Proposition 4. *Let $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ and $\mathcal{P} = \{V^k\}_{k \in [1,t]}$ be a partition in special form of G. Then, (L, bag) is a path decomposition of G. Its path-width is $n - 1$ plus the maximum number of not fully matched vertices in some $G(V^1 \cup \cdots \cup V^{k-1})$ (for $k \in [2, t]$).*

Example 4. *Figure 3 illustrates the path-decomposition of a graph isomorphic to the first graph of Figure 1 using the given vertex partition $V^1, \ldots, V^5$. The list of bags reported below each vertex of the graph, defines the set of all bags containing that vertex.*

Notice that in the example we do not need a vertex labeled by 0 and that we have chosen the vertex partition as well as the edges in order to minimize the path-width. In the following we show how to do this in general.

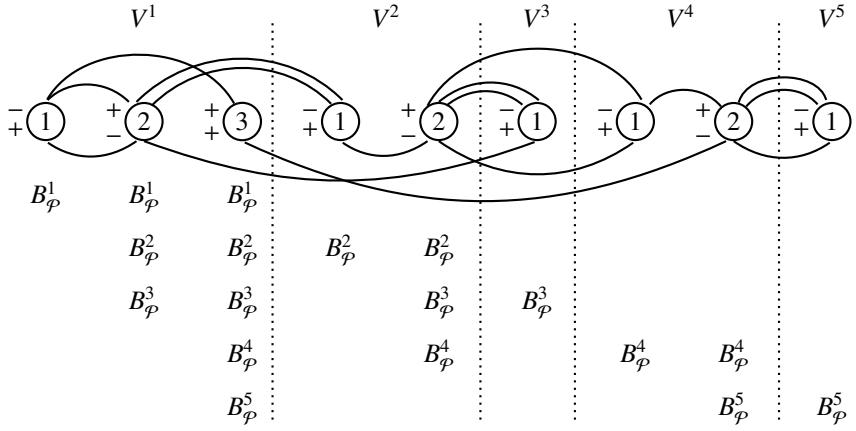

Figure 3: A path decomposition of the first graph in Figure 1 build from the vertex partition $\{V^1, \ldots, V^5\}$.

Definition 3. *Let $G \in \mathcal{G}[A\vec{x} = \vec{b}]$ and $\vec{s} = (s_1, \ldots, s_n)$ be a corresponding solution of $A\vec{x} = \vec{b}$. Let $\pi_C = S_1, \ldots, S_t$ be the sequence of sets of column indices of the $\vec{s}$-execution of* Balance*. Then, a partition $\mathcal{P} = \{V^k\}_{k\in[1,t]}$ in special form is called a* balanced vertex partition *iff V^1 contains exactly one vertex labeled by each $i \in [0, n]$, and for every $k > 0$, V^k contains exactly one vertex labeled by i for each $i \in S_k$.*

For the solution of the ILP instance of Example 1, the balanced vertex partition $\mathcal{P}$ is shown in Figure 3 by the sets $V^1, \ldots, V^5$.

5.2. Building a graph solution from a balanced vertex partition

Building on the balanced vertex partition $\mathcal{P} = \{V^k\}_{k\in[1,t]}$ given in Definition 3, we define one particular graph $G = (V, E, \{V_i\}_{i\in[0,n]}, \{E_j\}_{j\in[1,m]}) \in \mathcal{G}[A\vec{x} = \vec{b}]$ with $V = \bigcup_{k\in[1,t]} V^k$ whose path decomposition derived from $\mathcal{P}$ (see Section 5.1) has width bounded by $2 \cdot n$. This is proved exploiting Proposition 4. In particular, we show that it is possible to define the set of edges E such that for any $k \in [1, t]$ the maximum number of not fully matched vertices in $G(V^1 \cup \cdots \cup V^{k-1})$ is smaller than $n + 1$. $\mathcal{P}$ induces a partition of the set of edges E in any graph in $\mathcal{G}[A\vec{x} = \vec{b}]$. This edge-partition is $\{E^k\}_{k\in[1,t]}$ where $E^k = E \cap ((V^1 \cup \cdots \cup V^k) \times V^k)$. In the proof of Lemma 4 we inductively define $\{E^k\}_{k\in[1,t]}$, hence E, such that the bags $B^k_{\mathcal{P}}$ of the path decomposition of G built from $\mathcal{P}$ are small.

For each vertex $v \in (V^1 \cup \cdots \cup V^k)$ $(k > 0)$ labeled by i and each constraint j,

we define the number of *open edges* of v as:

$$open_{j,i}(v) = |a_{j,i}| - |\{(v, v') \in (E_j \cap (E^1 \cup \cdots \cup E^k))\}|.$$

We extend the definition of this function to a set of vertices as:

$$open_{j,i}(V^1 \cup \cdots \cup V^k) = \sum_{v \in (V^1 \cup \cdots \cup V^k) \cap V_i} open_{j,i}(v).$$

Lemma 4. *There is a graph $G = (V, E, \{V_i\}_{i\in[0,n]}, \{E_j\}_{j\in[1,m]}) \in \mathcal{G}[A\vec{x} = \vec{b}]$ with balanced vertex partition $\mathcal{P} = \{V^k\}_{k\in[1,t]}$ such that the induced edge partition $\{E^k\}_{k\in[1,t]}$ is such that each subgraph $(V^1 \cup \cdots \cup V^k, E^1 \cup \cdots \cup E^k)$ contains at most one vertex labeled by i that is not completely matched.*

Proof. It suffices to show that, for $k \in [1, t-1]$, the number of open edges $open_{j,i}(V^1 \cup \cdots \cup V^k)$ can be bounded by $|a_{j,i}|$ and that all of those come from the same vertex labeled by i. We show how to choose $\{E^k\}_{k\in[1,t-1]}$ such that $open_{j,i}(V^1 \cup \cdots \cup V^k) = |c^k_{j,i}|$ where the $c^k_{j,i}$ come from Lemma 2 which shows $|c^k_{j,i}| \leq |a_{j,i}|$. To show that inductively let us consider the situation at iteration $k+1$ after adding V^{k+1} to the subgraph induced by $V^1 \cup \cdots \cup V^k$ (but not yet E^{k+1} which we will define). V^k contains vertices corresponding to variables i which are in the corresponding S_k (for $k = 1$, V^k contains a vertex for each variable i). The number of open edges of variable i which we call $d^k_{j,i}$ is given by $d^k_{j,i} = c^k_{j,i} + a_{j,i}$ (or just $a_{j,i}$ for $k = 1$), if $i \in S_{k+1}$, else $d^k_{j,i} = c^k_{j,i}$. We have $\sum_{p\in pos_j} d^k_{j,p} + \sum_{p\in neg_j} d^k_{j,p} = \sum_{p\in pos_j} c^{k+1}_{j,p} + \sum_{p\in neg_j} c^{k+1}_{j,p}$ because of (a) and (b) of Lemma 2.

That means that before and after adding E^{k+1} the difference between "positive" and "negative" open edges is the same. Furthermore, $\sum_{p\in pos_j} d^k_{j,p} \geq \sum_{p\in pos_j} c^{k+1}_{j,p}$ and $\sum_{p\in neg_j} d^k_{j,p} \leq \sum_{p\in neg_j} c^{k+1}_{j,p}$ and due to (c), $|c^{k+1}_{j,i}|$ decreases w.r.t. $|c^k_{j,i}|$ for variables i not in S_{k+1}. Therefore, E^{k+1} can be defined such that the number of open "positive" edges and open "negative" edges decreases simultaneously to get to $c^{k+1}_{j,i}$ from $d^k_{j,i}$: we just match sufficiently many open "positive" edges with open "negative" edges with edges in E^{k+1} while taking care of the fact that the remaining open edges for a given variable all come from the same vertex. □

Figure 3 illustrates the way edges are chosen. Notice that the number of open edges corresponds exactly to the values of the $c^k_{j,i}$ computed in Section 4. The number of open edges in the end is 0. Lemma 4 together with Proposition 4 implies Lemma 3.

6. Automata construction for ILP

In this section, we show a direct automata construction from an ILP instance $A\vec{x} = \vec{b}$ such that the Parikh image of the automaton coincides with the set of solutions of the ILP instance. We call such machines ILP automata.

ILP automata model the executions of Algorithm Balance in Figure 2(b), i.e., each accepting run of such an automaton corresponds to a terminating execution of this algorithm, and vice-versa [4]. The states of an ILP automaton are tuples of integer numbers representing the possible values of the counters $\vec{r} = (r_1, \ldots, r_m)$. Each transition of the automaton corresponds to adding a column of the matrix A to the current set of counters $\vec{r}$. The automaton has exactly one final state which is the vector $\vec{b}$.

An ILP automaton has a bounded number of states because the counters $\vec{r}$ in the algorithm Balance are themselves bounded. For a tuple of natural numbers $\vec{r} = (r_1, r_2, \ldots, r_m)$, we say that $\vec{r}$ is *bounded* iff

$$2 \cdot \sum_{\substack{i \in [1,n], \\ a_{j,i} < 0}} (a_{j,i}) \;\leq\; r_j \;\leq\; 2 \cdot \Big(b_j + \sum_{\substack{i \in [1,n], \\ a_{j,i} > 0}} (a_{j,i})\Big)$$

for every $j \in [1, m]$. Let R_m be the set of all bounded m-tuples $\vec{r}$. The bounds on the counters $\vec{r}$ in the ILP automaton are twice the bounds in the Algorithm Balance because the latter hold only at the *end* of every iteration of the `for` loop. Formally,

Definition 4 (ILP Automata)**.** *Let $I \triangleq A\vec{x} = \vec{b}$ be an ILP instance with m constraints over the variables $V = \{x_1, \ldots, x_n\}$. The* ILP automaton *associated to I is the deterministic finite-state automaton $\mathcal{A}_I = (\Sigma, Q, q_0, F, \delta)$ defined as follows:*

- *the alphabet Σ is V;*
- *the set of states Q is R_m;*
- *the initial state $q_0 = \vec{0}_m$ is the m-tuple of 0s;*
- *the set of final states F is the singleton $\{\vec{b}\}$;*

[4] W.l.o.g. we assume that $\vec{b}$ is a tuple of non-negative integers.

- *the partial transition function* $\delta : Q \times \Sigma \mapsto Q$ *is defined as follows: let* $\vec{c_i}$ *be the i-th column of A. Then, for all* $x_i \in \Sigma$

$$\delta(\vec{r}, x_i) = \begin{cases} \vec{r} + \vec{c_i} & \text{if } \vec{r} + \vec{c_i} \text{ is bounded} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The following result is a direct consequence of Theorem 2.

Theorem 4. *Let* $I \triangleq A\vec{x} = \vec{b}$ *be an ILP instance and* $S_I \subseteq \mathbb{N}^n$ *the set of solutions of I. Then,* $Parikh(\mathcal{L}(\mathcal{A}_I)) = S_I$.

An interesting aspect of the automaton $\mathcal{A}_I$ is that it can be *implemented* as a compact Boolean program P_I whose size is linear in the size of I, as opposed to the exponential size of $\mathcal{A}_I$. P_I has a (bounded) variable r_j for each constraint. These variables are all initialized to zero. P_I iteratively guesses a symbol in Σ and updates the variables according to the transition function of $\mathcal{A}_I$. Now a special control location is reachable if and only if $\mathcal{A}_I$ accepts a word (when the tuple of constraint counters is $\vec{b}$). The intrinsic characteristic of P_I is that checking the reachability of the special location gives an answer to the ILP problem, and further this can be done with any verification tools designed for (Boolean) programs.

Notice that the automaton we define has a linear (in the number of variables) size alphabet and a state space of exponential size in the number of constraints. One can construct an automaton directly from the formula defined in Proposition 3 using the construction in [29] (Theorem 11.37) for finite satisfiability of FO formulae on graphs of bounded tree-width. However, the size of its alphabet is exponential in the number of variables and constraints and its state space is at least exponential in the size of the alphabet, i.e. of doubly exponential size. This holds because the formula in Proposition 3 on graphs has one quantifier alternation $\forall\exists$ and this automaton represents the models of an MSO formula describing path-decompositions of those graphs.

Besides the comparison in size, our automaton has the property that its Parikh image represents exactly the set of solutions of the ILP instance. This doesn't hold for an automaton obtained through the construction in [29].

7. Conclusion

In this paper we have investigated whether the intuition of interpreting ILP solutions with labelled graphs that are MSO definable and of bounded tree-width

also applies to the ILP feasibility problem. We have given a positive answer to this question showing that ILP feasibility can indeed be reduced in polynomial time to the satisfiability problem of FO (rather than MSO) on the class of bounded path-width (as opposed to bounded tree-width) graphs which is again decidable by Seese's theorem [5]. What we have not explored yet is whether our approach could also entail the optimal complexity of the problem. Although the ILP feasibility problem is NP-complete, the Boolean programs derived from the automata construction of Section 6 only lead to a PSPACE procedure. We believe it is interesting to shed some light in this regards. Furthermore, continuing the exploration in other directions by applying the approach of [2] and the one we propose in this paper to other decision problems is also an interesting venue for future research. For example, for several other classes of automata their decision procedures for the emptiness problem are derived by checking that the Parikh image of the language accepted by them satisfies a set of linear constraints (see for example [30]). We believe that combining the behaviour graphs of these automata with the solution graphs we proposed for ILP could lead to further applications of the approach to broader classes of automata.

References

[1] R. Alur, P. Madhusudan, Adding nesting structure to words, J. ACM 56 (3).

[2] P. Madhusudan, G. Parlato, The tree width of auxiliary storage, in: T. Ball, M. Sagiv (Eds.), POPL, ACM, 2011, pp. 283–294.

[3] H. L. Bodlaender, A tourist guide through treewidth, Acta Cybern. 11 (1-2) (1993) 1–22.

[4] B. Courcelle, The monadic second-order logic of graphs. i. recognizable sets of finite graphs, Inf. Comput. 85 (1) (1990) 12–75.

[5] D. Seese, The structure of models of decidable monadic theories of graphs, Ann. Pure Appl. Logic 53 (2) (1991) 169–195.

[6] M. F. Atig, A. Bouajjani, K. N. Kumar, P. Saivasan, Linear-time model-checking for multithreaded programs under scope-bounding, in: S. Chakraborty, M. Mukund (Eds.), ATVA, Vol. 7561 of Lecture Notes in Computer Science, Springer, 2012, pp. 152–166.

[7] S. La Torre, G. Parlato, Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width, in: D. D'Souza, T. Kavitha, J. Radhakrishnan (Eds.), FSTTCS, Vol. 18 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, pp. 173–184.

[8] S. La Torre, M. Napoli, A temporal logic for multi-threaded programs, in: J. C. M. Baeten, T. Ball, F. S. de Boer (Eds.), IFIP TCS, Vol. 7604 of Lecture Notes in Computer Science, Springer, 2012, pp. 225–239.

[9] S. Qadeer, J. Rehof, Context-bounded model checking of concurrent software, in: N. Halbwachs, L. D. Zuck (Eds.), TACAS, Vol. 3440 of Lecture Notes in Computer Science, Springer, 2005, pp. 93–107.

[10] S. La Torre, P. Madhusudan, G. Parlato, A robust class of context-sensitive languages, in: LICS, IEEE Computer Society, 2007, pp. 161–170.

[11] M. F. Atig, Model-checking of ordered multi-pushdown automata, Logical Methods in Computer Science 8 (3).

[12] S. La Torre, M. Napoli, Reachability of multistack pushdown systems with scope-bounded matching relations, in: J.-P. Katoen, B. König (Eds.), CONCUR, Vol. 6901 of Lecture Notes in Computer Science, Springer, 2011, pp. 203–218.

[13] S. La Torre, M. Napoli, G. Parlato, Scope-bounded pushdown languages, in: A. M. Shur, M. V. Volkov (Eds.), Developments in Language Theory - 18th International Conference, DLT 2014, Ekaterinburg, Russia, August 26-29, 2014. Proceedings, Vol. 8633 of Lecture Notes in Computer Science, Springer, 2014, pp. 116–128.

[14] S. La Torre, M. Napoli, G. Parlato, Scope-bounded pushdown languages, Int. J. Found. Comput. Sci. 27 (2) (2016) 215–234. doi:10.1142/S0129054116400074.
URL http://dx.doi.org/10.1142/S0129054116400074

[15] S. La Torre, M. Napoli, G. Parlato, A unifying approach for multistack pushdown automata, in: E. Csuhaj-Varjú, M. Dietzfelbinger, Z. Ésik (Eds.), Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I, Vol. 8634 of Lecture Notes in Computer Science, Springer, 2014, pp. 377–389. doi:10.1007/978-3-662-44522-8_32.

[16] A. Heußner, Model checking communicating processes: Run graphs, graph grammars, and mso, ECEASST 47.

[17] S. La Torre, P. Madhusudan, G. Parlato, Context-bounded analysis of concurrent queue systems, in: C. R. Ramakrishnan, J. Rehof (Eds.), TACAS, Vol. 4963 of Lecture Notes in Computer Science, Springer, 2008, pp. 299–314.

[18] A. Heußner, J. Leroux, A. Muscholl, G. Sutre, Reachability analysis of communicating pushdown systems, Logical Methods in Computer Science 8 (3).

[19] R. Parikh, On context-free languages, J. ACM 13 (4) (1966) 570–581.

[20] A. H. Land, A. G. Doig, An automatic method of solving discrete programming problems, Econometrica 28 (3) (1960) 497–520. doi:10.2307/1910129.

[21] R. E. Gomory, An algorithm for the mixed integer problem, Tech. rep., RAND Corporation (1960).

[22] A. K. Lenstra, H. W. L. Jr., L. Lovsz, Factoring polynomials with rational coefficients, Mathematische Annalen 261 (4) (1982) 515–534. doi:10.1007/BF01457454.

[23] W. Pugh, A practical algorithm for exact array dependence analysis, Commun. ACM 35 (8) (1992) 102–114.

[24] J. Büchi, Weak second-order arithmetic and finite automata., Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik 6 (1960) 66–92. doi:10.1002/malq.19600060105.

[25] V. Ganesh, S. Berezin, D. L. Dill, Deciding presburger arithmetic by model checking and comparisons with other methods, in: M. Aagaard, J. W. O'Leary (Eds.), FMCAD, Vol. 2517 of Lecture Notes in Computer Science, Springer, 2002, pp. 171–186.

[26] P. Wolper, B. Boigelot, An automata-theoretic approach to presburger arithmetic constraints (extended abstract), in: A. Mycroft (Ed.), SAS, Vol. 983 of Lecture Notes in Computer Science, Springer, 1995, pp. 21–32.

[27] C. H. Papadimitriou, On the complexity of integer programming, J. ACM 28 (4) (1981) 765–768.

[28] C. Enea, P. Habermehl, O. Inverso, G. Parlato, On the path-width of integer linear programming, in: A. Peron, C. Piazza (Eds.), Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014., Vol. 161 of EPTCS, 2014, pp. 74–87. doi:10.4204/EPTCS.161.9.
URL http://dx.doi.org/10.4204/EPTCS.161.9

[29] J. Flum, M. Grohe, Parameterized Complexity Theory, Texts in Theoretical Computer Science, Springer, 2006.
URL http://books.google.it/books?id=VfJz6hvFAjoC

[30] J. Esparza, P. Ganty, R. Majumdar, A perfect model for bounded verification, in: LICS, IEEE, 2012, pp. 285–294.