On the Path-Width of Integer Linear Programming

Constantin Enea Peter Habermehl
LIAFA, University of Paris Diderot and CNRS
75205 Paris 13
France
{cenea, Peter.Habermehl}@liafa.univ-paris-diderot.fr

Omar Inverso Gennaro Parlato
School of Electronics and Computer Science
University of Southampton
United Kingdom
{oi2c11,gennaro}@ecs.soton.ac.uk

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.

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.

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 [19], 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 charactering 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 [4]. The problem of MSO satisfiability on graphs is undecidable in general, but decidable on the class of bounded tree-width graphs [6, 24].

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 [19] 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 [19], model-checking of LTL [3, 16], and *generalised* LTL [15] for various restrictions of multi-stack PDA [23, 12, 2, 14], and (2) the reachability problem [19, 10] for subclasses of distributed automata that communicate through unbounded queues [13, 11]. 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 interesting 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 automaton A_I over the alphabet of I's variables, such that the Parikh image [21] 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

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

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 Sec. 2, we give basic definitions on graphs, tree-width, MSO on graphs, and the feasibility problem of ILP. In Sec. 3, we present the graph representation for ILP solutions, and give its FO characterisation. In Sec. 4, we give the bounded path-width theorem, and in Sec. 5 we describe the automata for ILP. We conclude with some remarks and future work in Sec. 6.

Related Work. Many approaches are known for solving the ILP feasibility problem, based on, e.g., branch-and-bound [17], the cutting-plane method [9], the LLL algorithm [18], the Omega test [22], finite-automata theory [5, 8, 25]. 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 [20] 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.

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: Fix two disjoint finite alphabets Σ_V and Σ_E. 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:

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

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 $\mathscr{C}$ is *MSO-definable*, resp., *FO-definable*, if there is an *MSO*, resp., *FO*, formula φ such that $\mathscr{C}$ 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 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 $u,v \in bag(n)$.
- 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 two leaves).

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 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 [24])**.** *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 $\mathscr{C}$ 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 $G \in \mathscr{C}$ of tree-width at most k that satisfies φ, is decidable.*

Integer Linear Programming (ILP): An *ILP instance* is a set 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 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 Graph representation for ILP solutions

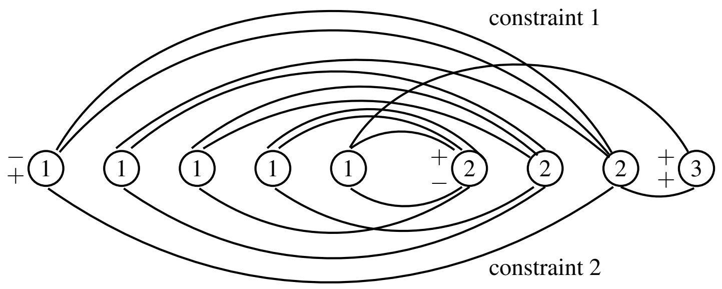
Given an ILP instance $A\vec{x} = \vec{b}$, we define the set of graphs $\mathscr{G}[A\vec{x} = \vec{b}]$ having the property that each graph in $\mathscr{G}[A\vec{x} = \vec{b}]$ represents a solution of $A\vec{x} = \vec{b}$. On the other hand, for every solution of $A\vec{x} = \vec{b}$ there is at least one graph in $\mathscr{G}[A\vec{x} = \vec{b}]$ representing it (but possibly more than one). Furthermore, we show that $\mathscr{G}[A\vec{x} = \vec{b}]$ 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 $\mathscr{G}[A\vec{x} = \vec{b}]$, 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

[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.

[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.

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, since the end-points of vertices represent the constants of the matrix A in unary, we can do the arithmetic related to each constraint 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 $a_{j,1} \cdot x_1 + \ldots + a_{j,n} \cdot x_n = b_j$ 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 $\mathscr{G}[A\vec{x} = \vec{b}]$ representing it. We now provide an example to illustrate this intuition.



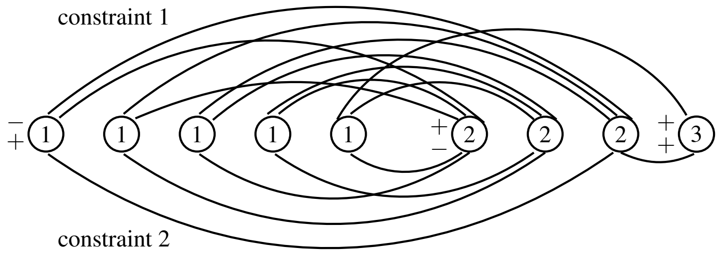


Figure 1: Two graph representations for the solution $x_1 = 5$, $x_2 = 3$, $x_3 = 1$ of $-2x_1 + 3x_2 + x_3 = 0$ and $x_1 - 2x_2 + x_3 = 0$. The edges above, resp., below, the vertices correspond to the first, resp, the second, equation. The signs attached to the vertices are the signs of the corresponding 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* $-2x_1 + 3x_2 + x_3 = 0$ *and* $x_1 - 2x_2 + x_3 = 0$.

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

Proposition 1. *The image of the function $sol:\mathscr{G}[A\vec{x}=\vec{b}]\to\mathbb{N}^n$ 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 $\mathscr{G}[A\vec{x}=\vec{b}]$. We show that for every $j\in[1,m]$, $sol(G)=(|V_1|,\dots,|V_n|)$ is a solution of the equation $a_{j,1}\cdot x_1+\ldots+a_{j,n}\cdot x_n=b_j$. Let $a_{j,0}=-b_j$. The set of indices $[0,n]$ can be partitioned in two sets $\{p_1,\dots,p_s\}$ and $\{n_1,\dots,n_t\}$ 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

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

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

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 $\mathscr{G}[A\vec{x}=\vec{b}]$ 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 $a_{j,1}\cdot x_1+\ldots a_{j,n}\cdot x_n=b_j$ we consider the partition of $[0,n]$ into $\{p_1,\dots,p_s\}$ and $\{n_1,\dots,n_t\}$ 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

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

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 $\mathscr{G}[A\vec{x}=\vec{b}]$ is non-empty.*

The following result shows that the class of graphs $\mathscr{G}[A\vec{x}=\vec{b}]$ 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, $G \in \mathscr{G}[A\vec{x} = \vec{b}]$ 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*, *Opposite*, and *Degree*, which express condition (1), (3), and (4) in Def. 1, respectively.

The condition on the vertex labels is given by the following formula:

$$\begin{aligned} \mathit{VertexLabels} \;\triangleq\; & \forall u. V_0(u) \oplus V_1(u) \oplus \ldots \oplus V_n(u) \\ & \wedge \exists v. V_0(v) \wedge \forall w, w'. \big((V_0(w) \wedge V_0(w')) \rightarrow w = w'\big), \end{aligned}$$

where $\oplus$ is the exclusive disjunction.

The formula *Opposite* is defined by:

$$\mathit{Opposite} \triangleq \forall u, v. \bigwedge_{j \in [1,m]} \Big(E_j(u,v) \rightarrow \mathit{opposite}_j(u,v)\Big)$$

where $\mathit{opposite}_j(u,v)$ 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. Formally, for any $j \in [1,m]$, let pos_j be the set of i such $a_{j,i} \geq 0$ 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 $b_j < 0$. Then,

$$\mathit{opposite}_j(u,v) \triangleq \left(\bigvee_{i \in \mathit{pos}_j} V_i(u) \wedge \bigvee_{i \in \mathit{neg}_j} V_i(v) \right) \vee \left(\bigvee_{i \in \mathit{neg}_j} V_i(u) \wedge \bigvee_{i \in \mathit{pos}_j} V_i(v) \right).$$

To express the constraint on the number of incident edges in a vertex of the graph, we introduce predicates of the form $E_j^k(u,v)$ with $k \in \mathbb{N}^*$, which holds iff there are exactly k edges labelled by j between u and v. Let *max* be the maximum value in A or $\vec{b}$, in absolute value. Then,

$$\mathit{Degree} \triangleq \underbrace{\forall u, v. \bigwedge_{j \in [1,m]} E_j(u,v) \rightarrow \big(E_j^1(u,v) \oplus \ldots \oplus E_j^{\mathit{max}}(u,v)\big)}_{\psi_1} \wedge \underbrace{\forall u. \bigwedge_{\substack{i \in [0,n] \\ j \in [1,m]}} \mathit{degree}_{i,j}}_{\psi_2}$$

where the sub-formula ψ_1 expresses the fact that, for any u and v, there exists exactly one predicate $E_j^k(u,v)$ which holds and $\mathit{degree}_{i,j}$ in ψ_2 is defined by:

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

Above, $\mathit{distinct}(u_1, \ldots, u_z)$ is the conjunction of all $u_i \neq u_{i'}$ with $1 \leq i \neq i' \leq z$. □

4 Bounded Path-width

In this section we show that for each solution $\vec{s}$ of $A\vec{x} = \vec{b}$ there is always a path-like graph representation in $\mathscr{G}[A\vec{x} = \vec{b}]$. More precisely, we show that for any solution of an ILP instance with n variables, there is a graph representation of this solution of path-width $2n$. Thus, the decidability of the ILP feasibility problem can be directly derived by the decidability of FO on the class of bounded path-width graphs.

Lemma 1. *For each solution $\vec{s}$ of $A\vec{x} = \vec{b}$ with n variables, there is a graph $G \in \mathscr{G}[A\vec{x} = \vec{b}]$ whose path-width is upper-bounded by* $2n$.

Proof. For each solution $\vec{s}$, $\mathscr{G}[A\vec{x} = \vec{b}]$ 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. Here, we show that among all graphs in $\mathscr{G}[A\vec{x} = \vec{b}]$ representing $\vec{s}$, there exists one whose path-width is bounded by $2n$. Without loss of generality we assume that $\vec{s}$ is a reduced solution, i.e. it is not a multiple of another solution and that $\vec{s}$ does not contain 0.

Let $\vec{s} = (s_1, \ldots, s_n)$. The proof is given by fixing $\vec{b} = \vec{0}$. In this case the path-width can be bounded by $2n-1$. At the end we show how to generalize the proof to any $\vec{b}$. Any graph $G \in \mathscr{G}[A\vec{x} = \vec{b}]$ with $sol(G) = \vec{s}$ has s_i vertices labelled by i. We say that $G \in \mathscr{G}[A\vec{x} = \vec{b}]$, with $G = (V, E, \{V_i\}_{i\in[0,n]}, \{E_j\}_{j\in[1,m]})$, is in *special form* if it satisfies the following two conditions: (1) there is a partition $\{V^k\}_{k\in[1,t]}$ of V such that no two vertices of V^k are labelled with the same variable index (i.e. $|V^k \cap V_i| \leq 1$ for all i,k), and (2) there is a partition $\{E^k\}_{k\in[1,t]}$ of E such that for all k, all edges in $E^1 \cup \ldots \cup E^k$ only relate vertices in $V^1 \cup \ldots \cup V^k$.

We now define a path decomposition for G in special form. We say that a vertex v in V in a subgraph of G is fully matched if the subgraph contains all edges of G involving v. The bags of the path decomposition of G are given by the sequence $B_p^0, B_p^1, B_p^2, \ldots, B_p^t$. Initially, $B_p^0 = \emptyset$. At each step k, $B_p^{k+1} = B_p^k \setminus \{v \in B_p^k \mid v \text{ is fully matched in } (V^1 \cup \cdots \cup V^k, E^1 \cup \cdots \cup E^k)\} \cup V^{k+1}$.

It is clear that the linear graph whose vertices are the bags $B_p^1, \ldots, B_p^k$ is a path decomposition for G. The size of the bags depends on the particular partition of vertices and edges.

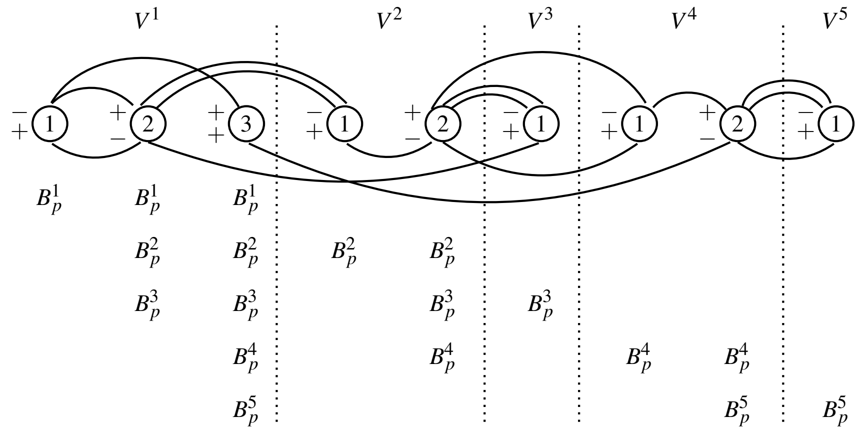


Figure 2: A path decomposition of the first graph in Figure 1

In Figure 2 we give the graph and its path decomposition computed by our algorithm explained below for the ILP instance in Example 1. The sets $V^1, \ldots, V^5$ are indicated by the dotted lines. For each vertex, the bags $B_p^1, \ldots, B_p^5$ to which it belongs to are indicated below it. Notice that the graph in the figure is isomorphic to the first graph in Figure 1.

In the following, we will define $\{V^k\}_{k\in[1,t]}$ and $\{E^k\}_{k\in[1,t]}$ such that the sizes of the bags are bounded by $2n$. The idea is to pick the V^k in a particular order such that it is always possible to add edges making sufficiently many vertices fully matched which allows to drop them from the corresponding bag. We will show that it is always possible to have at most 2 vertices labelled by the same variable in each bag.

We will give now an auxiliary algorithm allowing us to define the partition $\{V^k\}_{k\in[1,t]}$. Consider two sets of counters $c_1,\dots,c_n$ and $r_1,\dots,r_m$ associated with the matrix A in a way that we explain below. Let $s_l = max(s_1,\dots,s_n)$. Initially $\forall i.c_i = 0$. We define two possible actions on c_i:

INCREASE(i)**:** performs $c_i = c_i + s_l$;

REDUCE$()$**:** performs $c_i = c_i - s_i$, $\forall i \in [1,n]$.

When c_i changes, all the counters r_j are updated to $\sum_i((\text{number of INCREASE}(i))\cdot a_{j,i})$. It is clear that, if we perform exactly s_i times INCREASE(i) for each $i\in[1,n]$ and s_l times REDUCE$()$, all the counters reach zero. The meaning of the counters c_i is purely functional to the algorithm we show below. The purpose of the counters r_j is to tell how far (in the j-th constraint) the solution is when the current assignment of the variable x_i is set to the number of INCREASE(i). When $r_j = 0$, the j-th constraint is satisfied.

Given the above mechanism, the counters c_i and r_j will range within a bounded interval if we use the following algorithm to determine the exact sequence of steps to perform:

1. INCREASE(i) while there is some i such that $c_i < s_i$
2. REDUCE$()$ and stop if $\forall i.c_i = 0$
3. goto (1.)

It is easy to see that for all counters c_i we have $0 \le c_i < 2\cdot s_l$ and after reduce steps $0 \le c_i \le s_l$. For the solution of the ILP instance of Example 1 the sequence of counter values (c_1,c_2,c_3) computed before and after each of the five REDUCE$()$ steps is $(0,0,0) \to \cdots (5,5,5) \to_{\text{R()}} (0,2,4) \to \cdots (5,7,4) \to_{\text{R()}} (0,4,3) \to (5,4,3) \to_{\text{R()}} (0,1,2) \to \cdots (5,6,2) \to_{\text{R()}} (0,3,1) \to (5,3,1) \to_{\text{R()}} (0,0,0)$. Similarly, the sequence of counter values (r_1,r_2) at each reduce step is $(2,0),(3,-5),(1,0),(2,-5),(0,0)$.

Now we prove by induction on the number of steps that

$$r_j = \frac{c_1 a_{j,1} + \cdots + c_n a_{j,n}}{s_l}. \tag{1}$$

Trivially the property holds at the beginning as all counters c_i are set to 0.

If the k-th step is INCREASE(i), this new value will be:

$$r_j + a_{j,i} = r_j + \frac{s_l}{s_l} a_{j,i} = \frac{c_1 a_{j,1} + \cdots + (c_i + s_l) a_{j,i} + \cdots + c_n a_{j,n}}{s_l} = r'_j.$$

If the k-th step is REDUCE$()$, then:

$$r'_j = \frac{(c_1 - s_1) a_{j,1} + \cdots + (c_n - s_n) a_{j,n}}{s_l} = r_j - \frac{s_1 a_{j,1} + \cdots + s_n a_{j,n}}{s_l} = r_j$$

(note that REDUCE$()$ steps do not affect the counters r_j).

This proves expression (1). Furthermore, since $\frac{c_i}{s_l} < 2$, we have:

$$|r_j| = |\frac{c_1 a_{j,1}}{s_l} + \cdots + \frac{c_n a_{j,n}}{s_l}| < 2\cdot n \cdot max_i |a_{j,i}|$$

which gives an upper bound on the absolute value of the counters $r_1,\dots,r_m$.

We define now the partition $\{V^k\}_{k\in[1,t]}$ (where t is the number of REDUCE steps) by taking as V^1 a set of vertices containing exactly one vertex labelled by each $i\in[1,n]$ and as V^k (for $k>1$) a set of vertices containing exactly one vertex labelled by i for each INCREASE(i) operation done between the k-th and $(k+1)$-th REDUCE step.

Now, it remains to define the partition of edges $\{E^k\}_{k\in[1,t]}$. First we define for each vertex v labelled by i of the set $V^1\cup\cdots\cup V^k$ ($k\geq 1$) and each constraint j the number of *open* edges. Let $open_{j,i}(v) = |a_{j,i}| - |\{(v,v')\in E_j\cap(E^1\cup\cdots\cup E^k)\}|$. Then, $open_{j,i}(V^1\cup\cdots\cup V^k) = \sum_{v\in(V^1\cup\cdots\cup V^k)\cap V_i} open_{j,i}(v)$. We will show that the number of open edges $open_{j,i}(V^1\cup\cdots\cup V^k)$ can be bounded by $|a_{j,i}|$ for each k. That means that each subgraph $(V^1\cup\cdots\cup V^k, E^1\cup\cdots\cup E^k)$ contains at most one vertex labelled by i not completely matched. That in turn means that B^k_p never contains more than 2 vertices labelled by i, since B^k_p is composed of all vertices of $V^1\cup\cdots\cup V^k$ not completely matched as well as all vertices of V^k (which contains at most one vertex for each variable).

We first define, from the sequence of values $c^1_j,\ldots,c^t_j$ of c_j after each reduce step for each variable i, a sequence $c^1_{j,i},\ldots,c^t_{j,i}$ of integers. These integers will indicate the number of open edges for each type of vertex after each reduce step (the number is positive or negative depending on the sign of $a_{j,i}$). Let $r^1_j,\ldots,r^t_j$ be the sequence of values of the counter r_j after reduce steps. We define the sets $pos_j = \{i \mid a_{j,i}\geq 0\}$ and $neg_j = \{i \mid a_{j,i} < 0\}$. Then, for each $k\in[1,t]$ we define for each value r^k_j its positive part $r^k_{j,pos} = (\sum_{p\in pos_j}(c^k_p\cdot a_{j,p}))/s_l$ and its negative part $r^k_{j,neg} = (\sum_{p\in neg_j}(c^k_p\cdot a_{j,p}))/s_l$ such that $r^k_j = r^k_{j,pos}+r^k_{j,neg}$. In the example we have the following successive values for the $(r^k_{1,pos}, r^k_{2,pos})$: $(2,\frac{4}{5}),(3,\frac{3}{5}),(1,\frac{2}{5}),(2,\frac{1}{5}),(0,0)$ and for $(r^k_{1,neg}, r^k_{2,neg})$: $(0,-\frac{4}{5}),(0,-\frac{8}{5}),(0,-\frac{2}{5}),(0,-\frac{6}{5}),(0,0)$. Now, it is easy to see that we can choose $c^k_{j,i}\in\{\lfloor\frac{a_{j,i}c^k_i}{s_l}\rfloor,\lceil\frac{a_{j,i}c^k_i}{s_l}\rceil\}$ such that (a) $\sum_{p\in pos_j} c^k_{j,p} = \lceil r^k_{j,pos}\rceil$, (b) $\sum_{p\in neg_j} c^k_{j,p} = \lfloor r^k_{j,neg}\rfloor$ and (c) $|c^k_{j,i}|\geq|c^{k+1}_{j,i}|$, if there was no INCREASE(i) operation between the k-th and the $(k+1)$-th REDUCE(). (a) and (b) guarantee $c^k_{j,1}+\cdots+c^k_{j,n} = r^k_j$. Furthermore, we have $|c^k_{j,i}|\leq|a_{j,i}|$, as $0\leq c^k_j\leq s_l$. In the example we choose as successive values for $(c^k_{1,1},c^k_{1,2},c^k_{1,3})$: $(0,2,0)$, $(0,3,0)$, $(0,1,0)$, $(0,2,0)$, $(0,0,0)$ and we choose as successive values for $(c^k_{2,1},c^k_{2,2},c^k_{2,3})$: $(0,-1,1)$, $(0,-2,1)$, $(0,-1,1)$, $(0,-2,1)$, $(0,0,0)$.

Now, we can show that we can choose $\{E^k\}_{k\in[1,t]}$ such that $open_{j,i}(V^1\cup\cdots\cup V^k) = |c^k_{j,i}|$. Furthermore, since $|c^k_{j,i}|\leq|a_{j,i}|$ we can always make sure that there is at most one not fully matched vertex for each variable i in $V^1\cup\cdots\cup V^k$. To show that inductively let us consider the situation just before the k-th REDUCE() step. V_k contains vertices corresponding to variables i with an INCREASE(i) operation after the $(k-1)$-th REDUCE() (for $k=1$, V_k contains a vertex for each variable i). The number of open edges (before adding E^k) of variable i which we call $d^{k-1}_{j,i}$ is given by $d^{k-1}_{j,i} = c^{k-1}_{j,i}+a_{j,i}$ (or just $a_{j,i}$ for $k=1$) for the vertices labelled by variable i for which an INCREASE(i) operation has been performed after the $(k-1)$-th REDUCE(); and the number of open edges is $d^{k-1}_{j,i} = c^{k-1}_{j,i}$ for the other variables i. We know that $\sum_{p\in pos_j} d^{k-1}_{j,p}+\sum_{p\in neg_j} d^{k-1}_{j,p}$ is equal to $\sum_{p\in pos_j} c^k_{j,p}+\sum_{p\in neg_j} c^k_{j,p}$ because of (a) and (b). That means that before and after a REDUCE() the difference between "positive" and "negative" open edges is the same. Furthermore $\sum_{p\in pos_j} d^{k-1}_{j,p}\geq\sum_{p\in pos_j} c^k_{j,p}$ and $\sum_{p\in neg_j} d^{k-1}_{j,p}\leq\sum_{p\in neg_j} c^k_{j,p}$ and due to (c), $|c^k_{j,i}|$ decreases w.r.t. $|c^{k-1}_{j,i}|$ for not increased variables. Therefore, E^k can be defined such that the number of open "positive" edges and open "negative" edges decreases simultaneously to get to $c^k_{j,i}$ from $d^{k-1}_{j,i}$. This concludes the proof for $\vec{b}=\vec{0}$.

If $\vec{b}\neq\vec{0}$, we just consider having an additional variable labelled by 0 with coefficients $a_{j,0} = -b_j$ (for $1\leq j\leq m$). The vertex labelled by 0 can be put into all V^k. The edges involving 0 are computed like the other edges. □

From Lemma 1 and Proposition 2 we get the following theorem.

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

From that we obtain the following corollary.

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 with path-width $2n$.*

5 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.

We reuse the ideas in the proof of Lemma 1 from Section 4 in order to build an automaton whose states are tuples of integer numbers representing the possible values of the counters $r_1, \dots, r_m$, paired with a bit $B \in \{0,1\}$. We can think of each accepting run of the automaton as a way of discovering a solution $(x_1 = s_1, \dots, x_n = s_n)$ for the ILP instance, starting with an initial assignment $(x_1 = 0, \dots, x_n = 0)$ and continuing by increasing exactly one x_i at each step. The run should also contain a step where the vector of coefficients $-\vec{b}$ is added to the current valuation of $r_1, \dots, r_m$. The bit B is used to ensure that $-\vec{b}$ is added exactly once. The way we have enumerated graph vertices in order to obtain path decompositions of bounded width defines also the manner in which to pick an x_i for the next increase such that the counters have bounded range (which implies that the state space is bounded). Formally,

Definition 2 (ILP AUTOMATA)**.** *Let $I \triangleq A\vec{x} = \vec{b}$ be an ILP instance over the variables $V = \{x_1, \dots, x_n\}$ and m constraints. The* ILP automaton *$\mathscr{A}_I$ associated to I is the DFA $(\Sigma, Q, \delta, s_0, F)$ defined as follows. For a tuple of natural numbers $\vec{r} = (r_1, r_2, \dots, r_m)$ we say that $\vec{r}$ is* bounded *iff $r_j \leq 2 \cdot n \cdot max_i |a_{j,i}|$ for every $j \in [1,m]$. Let R_m be the set of all bounded m-tuples $\vec{r}$. Then,*

- *$\Sigma = V \cup \{b\}$ is the alphabet of $\mathscr{A}_I$;*
- *$Q = (\{0,1\} \times R_m)$ is the set of states;*
- *the transition map $\delta : Q \times \Sigma \mapsto 2^Q$ is defined as follows. Let A_i be the i'th column of A. Then,*

$$\delta((B,\vec{r}),x) = \begin{cases} \{(B,\vec{r'}) \mid \vec{r'} = \vec{r} + A_i,\ \vec{r'} \text{ is bounded}\} & \text{if } x = x_i \text{ with } i \in [1,n]; \\ \{(1,\vec{r'}) \mid \vec{r'} = \vec{r} - \vec{b},\ \vec{r'} \text{ is bounded}\} & \text{if } x = b \text{ and } B = 0; \\ \emptyset & \text{otherwise.} \end{cases}$$

- *the initial state s_0 is the pair $(0, \vec{0}_m)$, where $\vec{0}_m$ is an m-tuple of 0's.*
- *the set of final states F is the singleton $\{(1, \vec{0}_m)\}$.* □

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

Theorem 3. *For any ILP instance $I \triangleq A\vec{x} = \vec{b}$, $\mathit{Parikh}(L(\mathscr{A}_I)) = S_I$, where $L(\mathscr{A}_I)$ is the language of $\mathscr{A}_I$ and $S_I \subseteq \mathbb{N}^n$ is the set of solutions of I.*

Proof. (Sketch) By the construction of $\mathscr{A}_I$, the Parikh image of any word accepted by the automaton is a solution of I. Now, given a solution $\vec{s}$ of I, take the sequence of steps $\textsc{increase}(i)$ and $\textsc{reduce}()$ used to define a path decomposition for the graph representation of $\vec{s}$ in Lemma 1. The projection of this sequence on the steps $\textsc{increase}(i)$ corresponds to an accepting run in the automaton $\mathscr{A}_I$ (each $\textsc{increase}(i)$ corresponds to a transition over the symbol x_i). $\square$

An interesting aspect of the automaton $\mathscr{A}_I$ 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 $\mathscr{A}_I$. P_I has a (bounded) variable r_i for each constraint, and a bit B to keep track of whether $\vec{b}$ has already been used. 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 $\mathscr{A}_I$. Now a special control location is reachable if and only if $\mathscr{A}_I$ accepts a word (when all constraint counters are 0 and B is set to 1). 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.

6 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 [24]. 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 5 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 [19] 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 is derived by checking that the Parikh image of the language accepted by them satisfies a set of linear constraints (see for example [7]). 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] Rajeev Alur & P. Madhusudan (2009): *Adding nesting structure to words*. *J. ACM* 56(3). Available at `http://doi.acm.org/10.1145/1516512.1516518`.

[2] Mohamed Faouzi Atig (2012): *Model-Checking of Ordered Multi-Pushdown Automata*. *Logical Methods in Computer Science* 8(3). Available at `http://dx.doi.org/10.2168/LMCS-8(3:20)2012`.

[3] Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar & Prakash Saivasan (2012): *Linear-Time Model-Checking for Multithreaded Programs under Scope-Bounding*. In Supratik Chakraborty & Madhavan Mukund, editors: *ATVA*, *Lecture Notes in Computer Science* 7561, Springer, pp. 152–166. Available at `http://dx.doi.org/10.1007/978-3-642-33386-6_13`.

[4] Hans L. Bodlaender (1993): *A Tourist Guide through Treewidth*. *Acta Cybern.* 11(1-2), pp. 1–22. Available at `http://www.inf.u-szeged.hu/kutatas/actacybernetica/vol11n12/bodlaen/bodlaen.xml`.

[5] J. Büchi (1960): *Weak second-order Arithmetic and Finite Automata.* Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik 6, pp. 66–92, doi:10.1002/malq.19600060105.

[6] Bruno Courcelle (1990): *The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs.* Inf. Comput. 85(1), pp. 12–75. Available at `http://dx.doi.org/10.1016/0890-5401(90)90043-H`.

[7] Javier Esparza, Pierre Ganty & Rupak Majumdar (2012): *A Perfect Model for Bounded Verification.* In: LICS, IEEE, pp. 285–294. Available at `http://dx.doi.org/10.1109/LICS.2012.39`.

[8] Vijay Ganesh, Sergey Berezin & David L. Dill (2002): *Deciding Presburger Arithmetic by Model Checking and Comparisons with Other Methods.* In Mark Aagaard & John W. O'Leary, editors: FMCAD, Lecture Notes in Computer Science 2517, Springer, pp. 171–186. Available at `http://dx.doi.org/10.1007/3-540-36126-X_11`.

[9] Ralph E. Gomory (1960): *An algorithm for the mixed integer problem.* Technical Report, RAND Corporation.

[10] Alexander Heußner (2012): *Model Checking Communicating Processes: Run Graphs, Graph Grammars, and MSO.* ECEASST 47. Available at `http://journal.ub.tu-berlin.de/eceasst/article/view/725`.

[11] Alexander Heußner, Jérôme Leroux, Anca Muscholl & Grégoire Sutre (2012): *Reachability Analysis of Communicating Pushdown Systems.* Logical Methods in Computer Science 8(3). Available at `http://dx.doi.org/10.2168/LMCS-8(3:23)2012`.

[12] Salvatore La Torre, P. Madhusudan & Gennaro Parlato (2007): *A Robust Class of Context-Sensitive Languages.* In: LICS, IEEE Computer Society, pp. 161–170. Available at `http://doi.ieeecomputersociety.org/10.1109/LICS.2007.9`.

[13] Salvatore La Torre, P. Madhusudan & Gennaro Parlato (2008): *Context-Bounded Analysis of Concurrent Queue Systems.* In C. R. Ramakrishnan & Jakob Rehof, editors: TACAS, Lecture Notes in Computer Science 4963, Springer, pp. 299–314. Available at `http://dx.doi.org/10.1007/978-3-540-78800-3_21`.

[14] Salvatore La Torre & Margherita Napoli (2011): *Reachability of Multistack Pushdown Systems with Scope-Bounded Matching Relations.* In Joost-Pieter Katoen & Barbara König, editors: CONCUR, Lecture Notes in Computer Science 6901, Springer, pp. 203–218. Available at `http://dx.doi.org/10.1007/978-3-642-23217-6_14`.

[15] Salvatore La Torre & Margherita Napoli (2012): *A Temporal Logic for Multi-threaded Programs.* In Jos C. M. Baeten, Thomas Ball & Frank S. de Boer, editors: IFIP TCS, Lecture Notes in Computer Science 7604, Springer, pp. 225–239. Available at `http://dx.doi.org/10.1007/978-3-642-33475-7_16`.

[16] Salvatore La Torre & Gennaro Parlato (2012): *Scope-bounded Multistack Pushdown Systems: Fixed-Point, Sequentialization, and Tree-Width.* In Deepak D'Souza, Telikepalli Kavitha & Jaikumar Radhakrishnan, editors: FSTTCS, LIPIcs 18, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 173–184. Available at `http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2012.173`.

[17] A. H. Land & A. G. Doig (1960): *An Automatic Method of Solving Discrete Programming Problems.* Econometrica 28(3), pp. 497–520, doi:10.2307/1910129.

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

[19] P. Madhusudan & Gennaro Parlato (2011): *The tree width of auxiliary storage.* In Thomas Ball & Mooly Sagiv, editors: POPL, ACM, pp. 283–294. Available at `http://doi.acm.org/10.1145/1926385.1926419`.

[20] Christos H. Papadimitriou (1981): *On the complexity of integer programming.* J. ACM 28(4), pp. 765–768. Available at `http://doi.acm.org/10.1145/322276.322287`.

[21] Rohit Parikh (1966): *On Context-Free Languages.* J. ACM 13(4), pp. 570–581. Available at `http://doi.acm.org/10.1145/321356.321364`.

[22] William Pugh (1992): *A Practical Algorithm for Exact Array Dependence Analysis.* Commun. ACM 35(8), pp. 102–114. Available at `http://doi.acm.org/10.1145/135226.135233`.

[23] Shaz Qadeer & Jakob Rehof (2005): *Context-Bounded Model Checking of Concurrent Software*. In Nicolas Halbwachs & Lenore D. Zuck, editors: *TACAS*, *Lecture Notes in Computer Science* 3440, Springer, pp. 93–107. Available at `http://dx.doi.org/10.1007/978-3-540-31980-1_7`.

[24] Detlef Seese (1991): *The Structure of Models of Decidable Monadic Theories of Graphs*. *Ann. Pure Appl. Logic* 53(2), pp. 169–195. Available at `http://dx.doi.org/10.1016/0168-0072(91)90054-P`.

[25] Pierre Wolper & Bernard Boigelot (1995): *An Automata-Theoretic Approach to Presburger Arithmetic Constraints (Extended Abstract)*. In Alan Mycroft, editor: *SAS*, *Lecture Notes in Computer Science* 983, Springer, pp. 21–32. Available at `http://dx.doi.org/10.1007/3-540-60360-3_30`.