Four Scenarios of Effective Computations on Sum-like Graphs

Elena V. Ravve and Zeev Volkovich

Department of Software Engineering
Ort Braude,
Karmiel, Israel
Email: {cselena,vlvolkov}@braude.ac.il

***Abstract*—In this paper, we consider computations on sum-like graphs, which we introduced in our previous works. For such graphs, we proposed a method that allows us to reduce the solution of a Monadic Second Order or First Order definable problem on the graphs to the solution of effectively derivable Monadic Second Order or First Order definable problems on their components, respectively. Now, we describe in great details four particular scenarios, where this method may be applied, and explain how it may improve the complexity of the solution. This lead us to a generalized formulation of Amdahl's style laws for each scenario. Moreover, we consider applications of our method to the fields of repetitive and hierarchical structures, widely used in hardware and software design, as well as parallel and distributed computations.**

***Keywords*—** *Sum-like graphs; Translation schemes; (Weighted) Monadic Second Order Logic ; First Order Logic; Repetitive and hierarchical structures; Incremental computations; Parallel and distributed computations.*

I. INTRODUCTION

Replacing of solution of a problem on a given graph by solution of other problems on derived graphs is widely used in different fields of science. In this paper, we address the case, when the original graph is *sum-like* and we extend the approach of [1], implemented in the framework of model checking. In [2], we describe in great details how the concept of FSM may be interpreted as a graph, and how computation of its properties may be expressed as a set of logical formulas. In [3], we investigate the case of computation with weighted automata on *sum-like labeled weighted trees*. In this paper, we are mostly concentrated on the complexity issues of the approach.

Assume we are given a model of an object in the form of graph G and a formalized presentation of a problem to be solved in the form of formula ϕ, possibly with free variables. Assume G is built from components G_i, where $i \in I$ is some index set or structure. Assume that G is a sum-like composition of G_i as defined in [1][2], or it is a sum-like labeled weighted tree, as defined in [3].

The main results of [1][2][3] show how the solution of ϕ on G depends on the components G_i of G and the index structure I. It is an extension of the Feferman–Vaught Theorem, cf. [4], for First Order Logic (FOL) to Monadic Second Order Logic ($MSOL$) or Weighted Monadic Second Order Logic ($WMSOL$). The detailed description of algorithmic use of Feferman–Vaught Theorem may be found in [5].

The Feferman–Vaught theorem covers a very wide class of generalized products and sums of structures and is extremely powerful. We extend these theorems to the case of (Weighted) Monadic Second Order Logic and it works only for a more restricted class of *sum–like* graphs (trees), cf. [1][2][3]. From our main theorems, we derive a method for solving $MSOL$ ϕ on sum-like G, which proceeds as follows (the similar treatment of $WMSOL$ ϕ on sum-like T see in [3]):

Preprocessing: Given ϕ and Φ, but no G, we construct *at once* a sequence of formulas $\psi_{\imath,\jmath}$ and a function $F_{\Phi,\phi}$. This construction is polynomial in the size of ϕ and Φ.

Incremental Computation: We compute the values $b_{\imath,\jmath}$ defined by $b_{\imath,\jmath} = 1$ iff $G_i \models \psi_{\imath,\jmath}$.

Final Integration: Our theorems now state that $G \models \phi$ iff $F_{\Phi,\phi}(\bar{b}) = 1$.

In this paper, we investigate how the fact that G is built from several components may be used in order to make the solution more effective in the general case as well as how the general approach is connected to different practical applications. In fact, this analysis gives a generalized formulation of Amdahl's style laws for each described computation model and scenario.

We consider four different scenarios, when the complexity gain may be reached. The first one, described in Section V, represents the case when our graph G is composed by repetition of some basic component(s) $G_\imath = \tilde{G}$. We may find lots of such constructions, for example, in chemistry: polymers, VLSI design: adders, shifters, memories and other applications, etc.

Very often, after the problem ϕ was solved once on G, it should be solved again on some light modification of G. The situation is investigated in Section VI. More precisely, let j denote the solution on the jth variant of G. We assume that G^j differs from G^{j+1} in one component ${G_i}^{j+1}$, otherwise we may consider a line of such variants $G^{j=j_0}, G^{j_1}, \ldots, G^{j_l=j+1}$. We consider two situations. The classical one looks at the cost of solving *once* the problem ϕ on G, and uses either the size of G, the size of ϕ or the sum of the two as the relevant input size. In addition, we ask, what our method can gain by repeating this process many times, with small changes at a time. For this purpose we also look at the size of the changed component and the number of iterations.

 ISBN: 978-1-61208-346-9

In Section VII, we consider the case, when computations may be done in parallel rather than sequentially on one computational unit. We show how our general approach leads us to the BSP computation model, as introduced in [6].

With the frequent use of the Internet, it becomes customary to have data distributed over many sites. We consider how our approach works in the case of the distributed databases in Section VIII. We show that under some reasonable conditions, our method leads to some variation of $LogP$ model, cf. [7].

The paper is structured as follows:

- We start from a list of notations.
- In Section II, we give a motivating example.
- In Section III, we recall general definitions and results taken almost verbatim from [2].
- Section IV provides detailed discussion of the common basis of all cost evaluations.
- In section V, we analyze complexity of single computations on repetitive structures.
- In section VI, we analyze complexity of incremental recomputations on the lightly modified graphs.
- In section VII, we analyze complexity of parallel computations.
- In section VIII, we analyze complexity of computations on distributed databases.
- Section IX summarizes the paper.

LIST OF NOTATIONS

$\pi_A R$	Projection of attributes A from relation R
$\sigma_\theta R$	Selection from R of tuples satisfying θ
$R \bowtie S$	Join of relations R and S
$b_{\imath,\jmath}$	Boolean values
$\bar{b}$	Vector of Boolean values
BSP	Bulk synchronous parallel model
$E_1, E_2, E_i, \ldots$	Sets of edges of a graph
$F_{\Phi,\phi}$	Computation, associated with Φ and ϕ
FOL	First Order Logic
FSM	Finite State Machine
$G, G_\Phi, \tilde{G}, G_\imath, \ldots$	Graph structures
I	Index structure
$I(\mathbf{R})\ldots$	Instances of Database schemes
$\mathcal{L}$	Logic
$MSOL$	Monadic Second Order Logic
$P_1, P_2, P_i, Q_1, Q_2, Q_i$	One place relations on the set of vertices
$\mathbf{R}, \mathbf{R}_I, \ldots$	Database schemes
$R_i, \ldots$	Relation symbols
$\rho(R_i)$	Arity of R_i
SOL	Second Order Logic
Φ	Translation scheme
$\Phi^*, \Phi^\#$	Two mappings of Φ
T	Sum-like tree
v	Vertex of a graph
$\bar{v}$	Vector of vertices of a graph
$V, V_1, V_2, V_i, V_\Phi, \ldots$	Sets of vertices of a graph
VLSI	Very Large Scale Integration
$WMSOL$	Weighted Monadic Second Order Logic

II. MOTIVATING EXAMPLE

In this section, we consider how verification of a $MSOL$–property over some composition of graphs can be reduced to its components. Assume we consider a city that is divided into two parts by a river, see Figure 1 taken from [8]. These parts are connected by bridges. Without lost of generality, let us assume that there exist only two bridges. Assume we are looking for one-way cycles in the city.

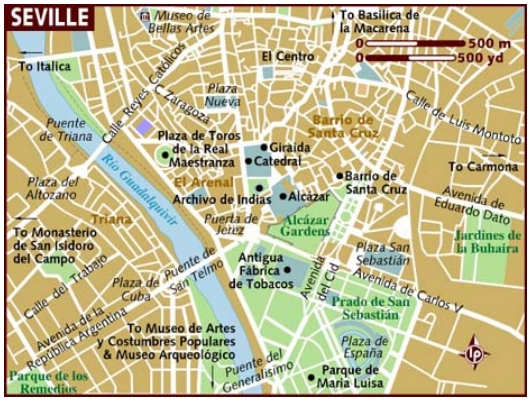

Fig. 1. A city with bridges [8].

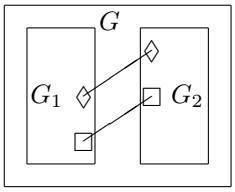

Fig. 2. Composition of two graphs: $G_1 \rightleftharpoons G_2$.

We may formulate the situation in the following way. We are given two finite graphs $G_1 = \langle V_1, E_1, P_1, Q_1\rangle$ and $G_2 = \langle V_2, E_2, P_2, Q_2\rangle$, where V_i denotes a set of vertices, E_i denotes a set of edges and P_i, Q_i are one place relations (vertex colorings) respectively. Let G be the disjoint union of G_1 and G_2 with additional edges, corresponding to the bridges, see Figure 2. We define this composition of two colored graphs formally as follows: $G = G_1 \rightleftharpoons G_2 = \langle V_1 \dot\cup V_2, E\rangle$, where $V_1 \dot\cup V_2$ denotes disjoint union of sets of vertices, and two vertices v and u of G belongs to E iff

$$\Phi : (v,u) \in E_1 \vee (v,u) \in E_2$$
$$\vee (v \in Q_1 \wedge u \in Q_2) \vee (v \in Q_2 \wedge u \in Q_1)$$
$$\vee (v \in P_1 \wedge u \in P_2) \vee (v \in P_2 \wedge u \in P_1)$$

We want to check whether G has cycles. To do so, we observe that

> (*) G has a cycle iff G_1 has a cycle, or G_2 has a cycle, or there are at least two connected coloured vertices in G_{2-i} and at least one coloured vertex in the same color vertex in G_{i+1}, where $i \in \{0,1\}$,

and proceed as follows.

- We first write the cyclicity property as a formula ϕ in $MSOL$.
- Then, using (*), which depends only on ϕ and Φ, but not G, we look for formulas $\psi_{1,1}, \ldots, \psi_{1,n_1}$ and

 ISBN: 978-1-61208-346-9

$\psi_{2,1}, \ldots, \psi_{2,n_2}$ in $MSOL$, which will give us the properties to be checked in G_1 and G_2 respectively.
- Then, again using (*), we look for a boolean function F of $n_1 + n_2$ arguments $b_{1,1}, \ldots, b_{2,n_2}$.
- Now we put $b_{i,j} = 1$ iff $G_i \models \psi_{i,j}$ and hope to conclude that $G \models \phi$ iff $F(b_{1,1}, \ldots, b_{2,n_2}) = 1$.

Surprisingly, our main theorems from [1][2][3] imply that this method can be mechanized in certain cost, even if (*) is not given in advance.

We have explained our main result by using a very simple example of a disjoint union of two colored graphs with some edges added, defined by Φ. The main theorems of [1][2][3] generalize this approach to combination of more than two structures and more complicated additional relations.

III. General Background

In this section, we recall general definitions and results taken almost verbatim from [2]. The corresponding extension to the case of $WMSOL$ may be found in [3].

A. Translation schemes

In this section, we follow [1] and introduce the general framework for syntactically defined translation schemes. A vocabulary is a finite set of relation symbols and constants.

Definition 1: General Translation Schemes.
Let τ and σ be two vocabularies and $\mathcal{L}$ be a logic, such as FOL or $MSOL$. Let $\sigma = \{R_1, \ldots, R_m\}$ and let $\rho(R_i)$ be the arity of R_i. Let $\Phi = \langle \phi, \psi_1, \ldots, \psi_m \rangle$ be formulas of $\mathcal{L}(\tau)$. Φ is *feasible for σ over τ* if ϕ has exactly 1 free first order variable and each ψ_i has $\rho(R_i)$ distinct free first order variables. Such a $\Phi = \langle \phi, \psi_1, \ldots, \psi_m \rangle$ is also called a *τ–σ–translation scheme* or, shortly, a *translation scheme*, if the parameters are clear in the context.

With a translation scheme Φ we can naturally associate a (partial) function Φ^* from τ–structures (graphs) to σ–structures (graphs).

Definition 2: The induced map Φ^*.
Let G be a τ–graph and Φ be feasible for σ over τ. The graph G_Φ is defined as follows:

1) The universe V_Φ of G_Φ is the set
$$V_\Phi = \{v \in V : G \models \phi(v)\};$$
2) The interpretation of R_i in G_Φ is the set
$$G_\Phi(R_i) = \{\bar{v} \in {G_\Phi}^{\rho(R_i)} : G \models \psi_i(\bar{v})\};$$
Note that G_Φ is a σ–graph of cardinality at most $\mid G \mid$.
3) The partial function $\Phi^* : \mathcal{G}(\tau) \to \mathcal{G}(\sigma)$ is defined by $\Phi^*(G) = G_\Phi$. Note that $\Phi^*(G)$ is defined iff $G \models \exists v \phi$.

With a translation scheme Φ we can also naturally associate a function $\Phi^\#$ from $MSOL(\sigma)$–formulas to $MSOL(\tau)$–formulas.

Definition 3: The induced map $\Phi^\#$.
Let θ be a σ–formula and Φ be feasible for σ over τ. The formula θ_Φ is defined inductively as follows:

1) For $R_i \in \sigma$ and $\theta = R_i(x_1, \ldots, x_m)$, we put $\theta_\Phi = \psi_i(v_1, \ldots, v_m)$.

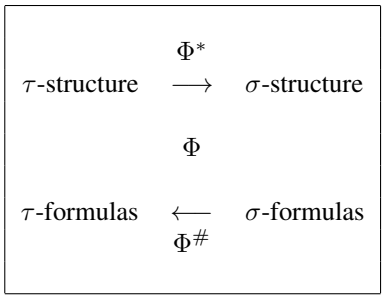

Fig. 3. Translation scheme and its components.

2) For the boolean connectives, the translation distributes, i.e., if $\theta = (\theta_1 \vee \theta_2)$ then $\theta_\Phi = (\theta_{1\Phi} \vee \theta_{2\Phi})$ and if $\theta = \neg\theta_1$ then $\theta_\Phi = \neg\theta_{1\Phi}$, and similarly for $\wedge$.
3) For the existential quantifier, we use relativization, i.e., if $\theta = \exists v \theta_1$, we put $\theta_\Phi = \exists v(\phi(v) \wedge (\theta_1)_\Phi)$.
4) For second order variables U of arity ℓ and u a vector of length ℓ of first order variables or constants we translate $\theta = \exists U \theta_1$, by treating U like a relation symbol and put $\theta_\Phi = \exists U(\forall u(U(u) \to (\phi(u_1) \wedge \ldots \phi(u_\ell) \wedge (\theta_1)_\Phi)))$.
5) The function $\Phi^\# : MSOL(\sigma) \to MSOL(\tau)$ is defined by $\Phi^\#(\theta) = \theta_\Phi$.

The following fact holds, see Figure 3:

Proposition 1:
Let $\Phi = \langle \phi, \psi_1, \ldots, \psi_m \rangle$ be a τ–σ–translation scheme, G a τ-graph and θ a σ–formula. Then $G \models \Phi^\#(\theta)$ iff $\Phi^*(G) \models \theta$. The proof may be found in [9][10].

B. Sum-like graphs

In this section, we discuss ways of obtaining graphs from components. The *Disjoint Union* of a family of graphs is the simplest example of juxtaposing graphs, where none of the components are linked to each other. For our purpose, we include the index set I in the resulting structure as well.

Definition 4: Disjoint Union of Graphs
Let $\tau_\imath = \langle R^\imath_1, \ldots, R^\imath_{j^\imath} \rangle$ be a vocabulary of graph $G_\imath$. In the general case the resulting graph $G = \dot{\bigcup}_{\imath \in I} G_\imath$ is
$G = \langle \mathcal{V} = I \cup \dot{\bigcup}_{\imath \in I} \mathcal{V}_\imath, R^I_j (1 \leq j \leq j^I), R^\imath_{j^\imath} (\imath \in I, 1 \leq j^\imath \leq j^\imath) \rangle$
for all $\imath \in I$, or rather any graph, isomorphic to it.
We assume existence of the following mappings:

- $h_\nu : \mathcal{V} \to I, h_\nu(v) = \imath$ if $v \in \mathcal{V}_\imath$;
- $h_\iota : PS(\mathcal{V}) \to PS(\mathcal{V}_\imath), h_\iota(V) = V_\imath$ if $V_\imath$ is a $\imath^{th}$ component of $\mathcal{V}$, while PS denotes the power set.

Definition 5: Partitioned Index Structure
Let I be an index structure. I is called *finitely partitioned* into ℓ parts if there are unary predicates I_α, $\alpha < \ell$, in the vocabulary of I such that their interpretation forms a partition of the universe of I.

The following holds:

Theorem 1:
Let I be a finitely partitioned index structure.
Let $G = \dot{\bigcup}_{i \in I} G_i$ be a τ–graph, where each G_i is isomorphic to some $B_1, \ldots, B_\ell$ over the vocabularies $\tau_1, \ldots, \tau_\ell$, in accor-

dance to the partition (ℓ is the number of the classes).
For every $\phi \in MSOL(\tau)$ there are:

- a boolean function $F_\phi(b_{1,1},\ldots,b_{1,j_1},\ldots,b_{\ell,1},\ldots,b_{\ell,j_\ell}, b_{I,1},\ldots,b_{I,j_I})$
- $MSOL$–formulas $\psi_{1,1},\ldots,\psi_{1,j_1},\ldots,\psi_{\ell,1},\ldots,\psi_{\ell,j_\ell}$
- $MSOL$–formulas $\psi_{I,1},\ldots,\psi_{I,j_I}$

such that for every G, I and $B_\imath$ as above with

$$B_\imath \models \psi_{\imath,\jmath} \text{ iff } b_{\imath,\jmath}=1 \text{ and } I \models \psi_{I,\jmath} \text{ iff } b_{I,\jmath}=1$$

we have

$$G \models \phi \text{ iff}$$

$$F_\phi(b_{1,1},\ldots,b_{1,j_1},\ldots,b_{\ell,1},\ldots,b_{\ell,j_\ell},b_{I,1},\ldots,b_{I,j_I})=1.$$

Moreover, F_ϕ and the $\psi_{\imath,\jmath}$ are computable from ϕ, ℓ and vocabularies alone, but are exponential in the quantifier depth of ϕ.

The disjoint union as such is not very interesting. However, combining it with translation schemes gives us a rich repertoire of patching techniques. Let τ_0,τ_1,τ be finite vocabularies of graphs. For a τ_0-model I (serving as index model), τ_1-graphs are pairwise disjoint for simplicity $G_\imath(\imath \in I)$ and a τ-graph G is the disjoint union of $\langle G_\imath : \imath \in I\rangle$ with $G=\dot{\bigsqcup}_{i\in I}G_i$. Now we generalize the disjoint union of graphs to *sum-like* graphs in the following way:

Definition 6: Sum–like Graphs
Let I be a finitely partitioned index structure.
Let $G=\dot{\bigsqcup}_{i\in I}G_i$ be a τ–graph, where each G_i is isomorphic to some $B_1,\ldots,B_\ell$ over the vocabularies $\tau_1,\ldots,\tau_\ell$, in accordance with the partition. Furthermore let Φ be a τ–σ $MSOL$–translation scheme. The Φ–sum of $B_1,\ldots,B_\ell$ over I is the graph $\Phi^*(G)$, or rather any graph isomorphic to it.

Theorem 2:
Let I be a finitely partitioned index structure and let G be the Φ–sum of $B_1,\ldots,B_\ell$ over I, as above.
For every $\phi \in MSOL(\tau)$ there are:

- a boolean function $F_{\Phi,\phi}(b_{1,1},\ldots,b_{1,j_1},\ldots,b_{\ell,1},\ldots,b_{\ell,j_\ell}, b_{I,1},\ldots,b_{I,j_I})$
- $MSOL$–formulas $\psi_{1,1},\ldots,\psi_{1,j_1},\ldots,\psi_{\ell,1},\ldots,\psi_{\ell,j_\ell}$
- $MSOL$–formulas $\psi_{I,1},\ldots,\psi_{I,j_I}$

such that for every G, I and $B_\imath$ as above with

$$B_\imath \models \psi_{\imath,\jmath} \text{ iff } b_{\imath,\jmath}=1 \text{ and } I \models \psi_{I,\jmath} \text{ iff } b_{I,\jmath}=1$$

we have

$$G \models \phi \text{ iff}$$

$$F_{\Phi,\phi}(b_{1,1},\ldots,b_{1,j_1},\ldots,b_{\ell,1},\ldots,b_{\ell,j_\ell},b_{I,1},\ldots,b_{I,j_I})=1.$$

Moreover, $F_{\Phi,\phi}$ and the $\psi_{\imath,\jmath}$ are computable from $\Phi^\#$ and ϕ, but are exponential in the quantifier depth of ϕ.

Moreover, in [3], we prove that:

Theorem 3:
Let I be a finitely partitioned index structure and a tree is the Φ–sum. For every $\phi \in WMSOL(\tau)$ there are:

- a computation on values $\varpi_{1,1},\ldots,\varpi_{\ell,j_\ell}$

$$F_{\Phi,\phi}(\varpi_{1,1},\ldots,\varpi_{1,j_1},\ldots,\varpi_{\ell,1},\ldots,\varpi_{\ell,j_\ell})$$

and

- $WMSOL$–formulas $\psi_{1,1},\ldots,\psi_{1,j_1},\ldots,\psi_{\ell,1},\ldots,\psi_{\ell,j_\ell}$

such that $\varpi_{\imath,\jmath}=\varrho_{\imath,\jmath}$ iff $[\psi_{\imath,\jmath}]=\varrho_{\imath,\jmath}$ we have

$$[\phi]=\varrho \text{ iff } F_{\Phi,\phi}(\varpi_{1,1},\ldots,\varpi_{1,j_1},\ldots,\varpi_{\ell,1},\ldots,\varpi_{\ell,j_\ell})=\varrho.$$

Moreover, $F_{\Phi,\phi}$ and the $\psi_{\imath,\jmath}$ are computable from $\Phi^\#$ and ϕ, but are exponential in the quantifier depth of ϕ.

IV. General Complexity Analysis

In this section, we discuss under what conditions theorems of [1][2][3] improve the complexity of computations, when measured in the size of the composed graphs (trees) only. Our scenarios are as follows: A $(W)MSOL$ formula (set of formulas) ϕ is given in advance. A sum-like graph (tree) is now submitted to a computation unit and we want to know, how long it takes to check whether ϕ is true on the graph (tree). Now we give the general complexity analysis of the computation on sum-like graphs (trees).

A. Complexity of computation for different logics

Theorems of [1][2] hold for $MSOL$ and, with restrictions, also for FOL. Computation for FOL is polynomial (even in logarithmic space), whereas computation for $MSOL$ is likely to be non–polynomial, as it sits fully in the polynomial hierarchy. Theorems of [3] hold for $WMSOL$. Computation for $WMSOL$ may be done, using Weighted Tree Automata.

More precisely, the complexity of computation (in the size of the graph) of Second Order Logic expressible properties can be described as follows. The class NP of non-deterministic polynomial-time problems is the set of properties, which are expressible by Existential Second Order Logic on finite structures, cf. [11]. Computation for SOL definable properties is in the polynomial hierarchy, cf. [12]. Moreover, for every level of the polynomial hierarchy there is a problem, expressible in SOL, that belongs to this class. The same fact hold for $MSOL$, too, as observed in [13].

Computation for properties, definable in Fixed Point Logic, is polynomial, cf. [14]. CTL^* is a superset of Computational Tree Logic and Linear Temporal Logic. All the problems, which are expressible by CTL^*, can be computed in polynomial time, cf. [15]. Most properties, which appears in real life applications, are stronger than FOL but weaker than $MSOL$, and their computational complexity is polynomial. In [16], it was shown that the similar theorems are valid for Transitive Closure and Monadic Fixed Point Logic. However, it does not hold for all of the languages (logics).

B. General analysis

Assume that G is a sum-like graph (or a sum-like tree). Its components are $G_\imath$ with index structure I, and we want to check whether ϕ is true in G. Assume that:

- $\mathcal{T}(N)$ or $\mathcal{T}_{old}(N)$ denotes time to solve the problem by the traditional sequential way (N denotes the size of the coding of graph G);
- $\mathcal{E}_\mathcal{I}$ denotes time to extract index structure I from G;
- $\mathcal{E}_\imath$ denotes time to extract each $G_\imath$ from G;

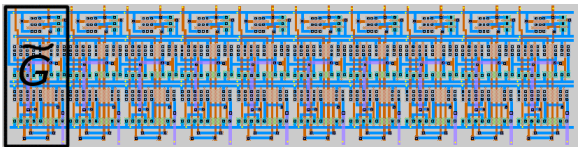

Fig. 4. Layout of a full 10-bit adder [22].

- $\mathcal{C}_{\mathcal{I}}(n_I)$ denotes time to compute all values of $b_{I,j}$, where n_I is the size of I;
- $\mathcal{C}_{\imath}(n_\imath)$ denotes time to compute all values of $b_{\imath,j}$, where $n_\imath$ is the size of $G_\imath$;
- $\mathcal{T}_{F_{\Phi,\phi}}$ denotes time to build $F_{\Phi,\phi}$;
- $\mathcal{T}_{\mathcal{S}}$ denotes time to achieve one result of $F_{\Phi,\phi}$.

According to these symbols, the new computation time is:

$$\mathcal{T}_{new} = \mathcal{E}_{\mathcal{I}} + \Sigma_{\imath \in I}\mathcal{E}_\imath + \mathcal{C}_{\mathcal{I}} + \Sigma_{\imath \in I}\mathcal{C}_\imath + \mathcal{T}_{F_{\Phi,\phi}} + \mathcal{T}_{\mathcal{S}} \quad (1)$$

and the question to answer is: when $\mathcal{T}_{old} > \mathcal{T}_{new}$.

V. Scenario A: Single Computation on Repetitive Structures

In this section, we consider the underlying structure, the formula and the modularity as well as possible applications of our method for single computations on repetitive structures. We analyze the corresponding complexity gain for the computations of properties, expressible in different logics.

A. The underlying structure, the formula and the modularity

Our underlying structure is a sum-like graph G, where for each $\imath$: $G_\imath = \tilde{G}$. The property, which we want to check on it, is expressible in formula ϕ of $MSOL$, which has an exponential checker. We check whether $G \models \phi$.

B. Applications

Different repetitive combinations of graphs have been under deep investigation for long time, cf. [17][18][19]. Polygraphs were introduced as generalization of polymers in chemistry, cf. [20], and explored in VLSI design, cf. [21].

We restrict ourselves to VLSI design, which widely uses repetition of blocks (Figure 4 taken from [22]) and hierarchical structures. Many basic elements of the design, such that shifters, adders, registers, etc. are build in this manner. Repetition of modules is explored also in control logic and other kinds of hardware design. Memory is another example of a repetitive structure in VLSI design. If we go up in the hierarchy of our design, we found multi-core processors, which are single computing components with two or more independent processors (called "cores").

C. Complexity gain for $MSOL$

Assume that our design is presented as a graph (FSM), such that:

- N is a size of G, n is a size of $\tilde{G}$ and l is a size of index structure I.
- The decomposition is given: $\mathcal{E}_{\mathcal{I}} = \mathcal{E}_\imath = 0$.
- The computation is exponential in the form: $\mathcal{T} = e^{g(x)}$.

In this case $\mathcal{T}_{new} = P^p(\mathcal{T}(n), \mathcal{T}(l))$, where P^p denotes polynomial of degree p, and $\mathcal{T}_{old} = \mathcal{T}(l \cdot n)$.

The question to answer is: when $f(n \cdot l) > P^p(f(n), f(l))$. According to our assumptions, we obtain that the comparison of the computation times in (1) looks like:

$$e^{g(n \cdot l)} > a_p(e^{p \cdot g(n)} + e^{p \cdot g(l)}).$$

Assume that $n = l$. Then $g(n^2) > p \cdot g(n) + ln2 + ln(a_p)$. Assume that $g(x) = ln^2(x)$, then $f(x) = x^{ln(x)}$. In this case we obtain that (1) is transformed to: $ln^2(n^2) > p \cdot ln^2(n) + ln2 + ln(a_p)$ or $ln^2(n) > \frac{ln(2 \cdot a_p)}{2-p}$.

D. Complexity gain for other logics

If we use FOL or another logic, where the computational procedure is polynomial in the sizes of G and in I and each $G_\imath$ too, then we do not obtain any time gain.

VI. Scenario B: Incremental Re-Computations

In this section, we consider the underlying structure, the formula and the modularity as well as possible applications of our method for incremental re-computations. We analyze the corresponding complexity gain for the computations of properties, expressible in different logics.

A. The underlying structure, the formula and the modularity

Our underlying structure is a sum-like graph, and the property, which we want to check on it, is expressible in a formula ϕ that has a polynomial checker. Assume that we change several times (let us denote the number of the times by ς) some fixed component $\breve{G}$ of G. We check each time whether $G \models \phi$.

B. Applications

We restrict ourselves to the case of hardware verification, where we find the following situation: We are given a mathematical model of a device in form of a finite relational structure G (FSM or Kripke model) and a formalized property ϕ. Usually ϕ is given in advance and G is being built with the aim to satisfy ϕ. Checking whether ϕ holds in G is to be atomized. This process is called model checking. The literature is rich in papers addressing this problem, cf. [15].

As a rule, hardware design G is built from components (modules) G_i, where $i \in I$. The modules are the building blocks, installing one in other that gives the design hierarchy. The communication between a module and its environment is executed using ports. All but the top-level modules in a hierarchy have ports. In the process of building G, several candidate structures G^j have to be checked for ϕ, where j denotes the j^{th} attempt of designing G. Often G^j differs from G^{j+1} in one component ${G_i}^{j+1}$. It is easy to see (Figure 2) that our motivating example exactly fails in this framework, when we are talking about hardware design rather than about city maps. Combination of graphs in terms of graph grammars may be found also in [23].

 ISBN: 978-1-61208-346-9

C. Complexity gain for FOL

Let $\mathcal{T}_{old}$ be time to solve the given problem by the traditionally applied way. It should be clear that $\mathcal{T}_{old} = \varsigma \cdot \mathcal{T}(N)$. Let $\mathcal{T}_{new}$ be time to solve the same problem, when structure G is viewed as a generalized sum. It is easy to see that

$$\mathcal{T}_{new}(N,n) = \mathcal{T}(N-n) + \varsigma \cdot \mathcal{T}(n) + \mathcal{T}_{F_{\Phi,\phi}} + \varsigma \cdot \mathcal{T}_{\mathcal{S}}.$$

The question to answer is: which value of n provides that $\mathcal{T}_{old} > \mathcal{T}_{new}$. Assume that $\mathcal{T}(x) = x^2$, then (1) becomes to be:

$$\varsigma \cdot N^2 > (N-n)^2 + \varsigma \cdot n^2 + \mathcal{T}_{F_{\Phi,\phi}} + \varsigma \cdot \mathcal{T}_S$$

$$N^2 - 2 \cdot n \cdot N + n^2(\varsigma+1) + \mathcal{T}_{F_{\Phi,\phi}} + \varsigma \cdot \mathcal{T}_S - \varsigma \cdot N^2 < 0$$

$$n_{1,2} = \frac{N \pm \sqrt{N^2 + (\varsigma+1)(N^2(\varsigma-1) - \mathcal{T}_{F_{\Phi,\phi}} - \varsigma \cdot \mathcal{T}_S)}}{\varsigma+1}.$$

If $n_1 \leq n \leq n_2$ then $\mathcal{T}_{old} > \mathcal{T}_{new}$.

$$n_2 = \frac{N + \sqrt{\varsigma^2(N^2 - \mathcal{T}_S) - \varsigma(\mathcal{T}_S + \mathcal{T}F_{\Phi,\phi}) - \mathcal{T}_{F_{\Phi,\phi}}}}{\varsigma+1}$$

$$lim_{\varsigma \to \infty} n_2 = \sqrt{N^2 - \mathcal{T}_S}.$$

The same consideration can be done for other polynomial dependencies $\mathcal{T}(x)$ for FOL-definable logics.

D. Complexity gain for other logics

Let $\mathcal{L}$ be any proper sub-logic of $MSOL$ stronger that FOL. Our theorems do not hold in the following: if we apply it then $\psi_{\imath,\jmath}$ are not necessary in $\mathcal{L}$.

VII. Scenario C: Parallel Computations

In this section, we consider the underlying structure, the formula and the modularity as well as possible applications of our method for parallel computations. We analyze the corresponding complexity gain for the computations of properties, expressible in different logics.

A. The underlying structure, the formula and the modularity

Our underlying structure is a sum-like graph G, and the property, which we want to check on the structure, is expressible in formula ϕ of a logic. We check whether $G \models \phi$.

B. Applications

Let us consider the following composition of two input graphs H and G. G can be viewed as a display graph, where on each node we want to have a copy of H, such that certain additional edges are added. In practice, this is a model on how a pipeline works. The nodes marked with L^j are the latches.

Let $G = \langle V_G, R\rangle$ and $H = \langle V_H, S, L^j(j \in J)\rangle$ be two relational structures (J is finite), then their composition $C = \langle V_C, L^1_C, ..., L^{|J|}_C, S_C, R^j_C(j \in J)\rangle$ is defined as follows, see Figure 5:

- $V_C = \dot{\bigcup}_{g \in G} V^g_H$, where each V^g_H is isomorphic to V_H;
- $L^j_C(w)$ is true if w belongs to L^j;
- $S_C = \{(w,v) : w \in V^g_H, v \in V^g_H, S(w,v)\}$;
- $R^j_C = \{(w,v) : L^j(w), L^j(v), P(i,w), P(i',v), R(i,i')\}$.

C can be obtained from the disjoint union $\dot{\bigcup}_{g \in G} H$ by a FOL translation scheme. In this example, depending on the choice of the interpretation of the L^j's, more sophisticated parallel computations can be modeled, but not all.

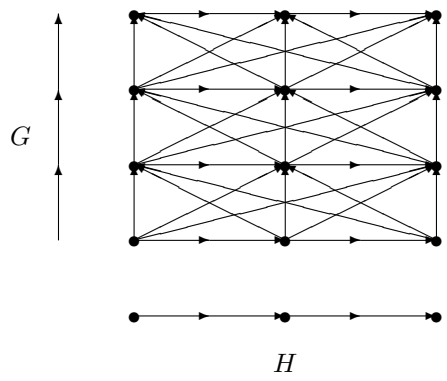

Fig. 5. Uniform graph substitution.

C. Complexity gain

In (1), the new computation time is calculated as:

$$\mathcal{T}_{new} = \mathcal{E}_{\mathcal{I}} + \Sigma_{\imath \in I}\mathcal{E}_\imath + \mathcal{C}_I + \Sigma_{\imath \in I}\mathcal{C}_\imath + \mathcal{T}_{F_{\Phi,\phi}} + \mathcal{T}_{\mathcal{S}}$$

for the case, when all the computations are done sequentially on a single computational unit. In fact, now even personal computers and smartphones have several cores. In this case, the computation may be done in the following way (we assume that there exist enough computational units for total parallelism) :

Extraction Super Step: The extraction of the index structure I from G and each $G_\imath$ from G may be done in parallel as well as the building of $F_{\Phi,\phi}$. We denote the extraction time by: $\mathcal{E} = max\{\mathcal{E}_I, max_{\imath \in I}\{\mathcal{E}_\imath\}, \mathcal{T}_{F_{\Phi,\phi}}\}$.

Computational Super Step: The computation of all values of $b_{I,\jmath}$ and $b_{\imath,\jmath}$ may be done in parallel as well. We denote by $\mathcal{C} = max\{\mathcal{C}_I(n_I), max_{\imath \in I}\{\mathcal{C}_\imath(n_\imath)\}\}$. In fact, at this step, even more parallelism may be reached if we compute all $b_{\imath,\jmath}$ in parallel.

Final Proceeding: $\mathcal{T}_{\mathcal{S}}$ still denotes time to search one result of $F_{\Phi,\phi}$. The new computation time for the case of full parallelism is: $\mathcal{T}^{BSP}_{new} = \mathcal{E} + \mathcal{C} + \mathcal{T}_{\mathcal{S}}$. The computation model fails in the general framework of BSP, cf. [6].

D. Complexity gain for $MSOL$

The computation is exponential in the form: $\mathcal{T} = e^{g(x)}$. In this case $\mathcal{T}_{old} = \mathcal{T} = f(N) = e^{g(N)}$ and $\mathcal{T}^{BSP}_{new} = \mathcal{E} + P^p(e^{g(\frac{N}{k})}) + \mathcal{T}_{\mathcal{S}}$ and the question to answer is: when $f(n \cdot k) > P^p(f(n))$. According to our assumptions, we obtain:

$$e^{g(n \cdot k)} > \mathcal{E} + a_p \cdot e^{p \cdot g(n)} + \mathcal{T}_{\mathcal{S}}.$$

Assume that $k = n$, it means that there exist enough computation units for full parallelization. In this case, the condition of the effective computation looks like:

$$e^{g(n^2)} > \mathcal{E} + a_p \cdot e^{p \cdot g(n)} + \mathcal{T}_{\mathcal{S}}.$$

 ISBN: 978-1-61208-346-9

E. Complexity gain for FOL and other logics with polynomial checkers

Assume that again $\mathcal{T}(x) = x^2$, and each G_i are of the same size $\frac{N}{k}$ then:

$$\mathcal{T}_{old} = N^2 \text{ and } \mathcal{T}^{BSP}_{new} = \mathcal{E} + (\tfrac{N}{k})^2 + \mathcal{T}_{\mathcal{S}}$$

$$N^2 > \mathcal{E} + (\tfrac{N}{k})^2 + \mathcal{T}_{\mathcal{S}} \; ; \; N^2 - (\tfrac{N}{k})^2 > \mathcal{E} + \mathcal{T}_{\mathcal{S}}$$

Now the condition of the effective computation looks like:

$$N^2 \cdot \tfrac{(k^2-1)}{k^2} > \mathcal{E} + \mathcal{T}_{\mathcal{S}}$$

Complexity consideration for other logics $\mathcal{L}$, which are proper sublogics of $MSOL$ stronger that First Order Logic, are similar to the given in subsection VI-D.

VIII. Scenario D: Parallel Computations on Distributed Data

In this section, we consider the underlying structure, the formula and the modularity as well as possible applications of our method for parallel computations on distributed databases. We analyze the corresponding complexity gain for the computations of queries, expressible in different logics.

A. The underlying structure, the formula and the modularity

Our underlying structure is a sum-like graph G that is stored in the distributed way: each G_i is stored in the i^{th} site. The property, which we want to check on the structure, is expressible in formula ϕ of a logic. We check whether $G \models \phi$.

B. Applications

We restrict ourselves to investigation of distributed databases. In this case, a user sees one (virtual) database instance over a fixed database scheme. Queries and updates are submitted to a central processing site which will compute the required view or transaction by distributing the appropriate sub-tasks among the different sites.

While Datalog is hard to check, cf. [14], in [24], a new variant of Datalog was introduced: Datalog LITE. On the one hand, the deductive query language has linear time model checking. On the other hand, it encompasses modal and temporal logics, such as CTL and alternation-free μ-calculus. Moreover, it was shown that Datalog LITE with only unary and binary input predicates is contained in $MSOL$.

Assume we are given a database scheme that contains four relations: $\mathbf{R} = (R_1, R_2, R_3, R_4)$. Assume that we want to define a view that is derived from the database by applying the following query, given in the format of relational algebra: $(\pi_A R_1 \cup R_2) \bowtie (R_3 - \sigma_\xi R_4)$. In this case, the corresponding translation scheme is: $\Phi_{View} = \langle x = x, \phi_{View} \rangle$, where $\phi_{View} = (\pi_A R_1 \cup R_2) \bowtie (R_3 - \sigma_\xi R_4)$.

Let $\mathbf{R}_I$ be an index scheme with finite domain and $|I| = n$ (to simplify the example, let n=2) and let $\mathbf{R}_1 = (R(y_1, \ldots, y_{r_1}))$, $\mathbf{R}_2 = (R(y_1, \ldots, y_{r_2}))$ be database schemes $\mathbf{R}_i (i \in \{1, 2\})$, where $r_j (j \in \{1, 2\})$ be an arity of the corresponding relation. Let $\mathbf{R}=\bigsqcup_{i \in I} \mathbf{R}_i = (P(i, x), I(x), R^1, R^2)$ be a $\mathbf{R}$ database scheme, which is the disjoint union of $\mathbf{R}_i$'s. We define the following translation scheme Φ_{Join} from the $\mathbf{R}$-instances to $\mathbf{S}$-instances, where $\mathbf{S} = (S)$.

$$\Phi_{Join} = \langle y \approx y, \exists y^1_{r_1} \exists y^2_1 (R^1(y^1_1, \ldots, y^1_{r_1}) \wedge$$
$$R^1(y^2_1, \ldots, y^2_{r_1}) \wedge \phi_{=}(y^1_{r_1}, y^2_1)) \rangle.$$

Assume we are given a sum-like database and we want to compute the view (query), defined by ϕ. Now, given a tuple t over $I(\mathbf{R})$, in order to check whether t belongs to the view, defined by ϕ, we compute the following:

1) $View^i_j = \{t \in I_i(\mathbf{R}) : \psi_j\}$, the views at site i, defined by the queries ψ_j.
2) $X^t_j = \{i \in I : t \in View^i_j\}$, the set of sites where t belongs to the view, defined by ψ_j;
3) The truth value of $I(\mathbf{R}_I) \models \psi_I(X^t_1, \ldots, X^t_n)$.

By our theorems: tuple t belongs to the view, defined by ϕ, iff $I(\mathbf{R}_I) \models \psi_I(X^t_1, \ldots, X^t_n)$. In other words

$$\{t : I(\mathbf{R}) \models \phi\} = \{t : I(\mathbf{R}_I) \models \psi_I(X^t_1, \ldots, X^t_n)\}.$$

We see that $View^i_j$ is computed at the site i and only the queries ψ_j have to be sent over the net. The $View^i_j$'s can be computed in parallel. Furthermore, when we compute X^t_j, only the tuple t is sent over the net. Finally, evaluating ψ_I can be done in $PSpace$ (in the size of I). However, it is likely that for special cases of ψ_I (when ϕ is a pure Datalog query) the complexity (in the size of I) becomes (at least non-deterministically) polynomial.

In distributed databases, where the data are measured in gigabytes (terabytes) and the size of I is a small finite set (say ≤ 100), this method, in spite of its considerable overhead, gives a considerable improvement over any other method, which moves parts of the databases over the net.

Further gains can be achieved by exploiting a hierarchical structure of the way the databases are distributed: we introduce virtual sites (gather sites), which gather the data of its sub-sites and make sure that the number of sub-sites remains bounded (say ≤ 10). At each gathering site, the evaluation of ψ_I remains thus feasible.

As it was shown in [25], our method generalizes the propagation technique from [26] for relational algebra and the incremental re-computation technique from [27] for some kinds of Datalog programs to cases of definable sets of tuples to be deleted or inserted.

In addition, assume that the query language allows us to ask optimization questions. In such cases, our generalized propagation technique may be directly extended to the case of incremental optimization as considered in [28][29][30]. Using our method, the final optimal result is computed from the local (not necessarily optimal) solutions as explained in [3]. Moreover, our approach is directly connected to Parallel Distributed Genetic Programming as introduced in [31].

C. Complexity gain

The full computation process is composed now from the following steps (the above $\mathcal{E} = 0$):

Computational Super Step The computation of all values of $b_{I,\jmath}$ and $b_{\imath,\jmath}$ is done in parallel in the corresponding sites. We still denote by $\mathcal{C} = max\{\mathcal{C}_I(n_I), max_{\imath\in I}\{\mathcal{C}_\imath(n_\imath)\}\}$. Recall that in each site, the $b_{\imath,\jmath}$ still may be computed in parallel if the corresponding computer has several cores.

Communication Super Step The results $b_{I,\jmath}$ and $b_{\imath,\jmath}$ must be sent for the final proceeding. We denote by $\mathcal{T}_I$ time to transfer all values of $b_{I,\jmath}$, and by $\mathcal{T}_\imath$ time to transfer all values of $b_{\imath,\jmath}$. The communication time now is calculated as $\mathcal{T} = max\{\mathcal{T}_I, max_{\imath\in I}\{\mathcal{T}_\imath)\}\}$.

Final Proceeding $\mathcal{T}_\mathcal{S}$ still denotes time to search one result of $F_{\Phi,\phi}$.

The new computation time of (1) for the case of the distributed storage and computation is: $\mathcal{T}^{distr}_{new} = \mathcal{C} + \mathcal{T} + \mathcal{T}_\mathcal{S}$. If the computations and the data transfer in each site may be done in parallel then we may combine two first super steps in the above model in one step that, in fact, leads to some variation of LogP model, introduced in [7]. If we denote by $\mathcal{D} = max\{(\mathcal{C}_I(n_I) + \mathcal{T}_I), max_{\imath\in I}\{(\mathcal{C}_\imath(n_\imath) + \mathcal{T}_\imath)\}\}$, then the corresponding computation time is: $\mathcal{T}^{LogP}_{new} = \mathcal{D} + \mathcal{T}_\mathcal{S}$.

IX. Conclusion

In this work, we consider computations on sum-like graphs. We considered different scenarios, when our method leads to improvement in the complexity of the computations. We have shown several applications of our method in the fields of design and verification of repetitive and hierarchical structures, parallel computations, computations on distributed data.

Each of the considered computational models may be combined with each of scenarios A-D in order to analyze $\mathcal{T}_{old}/\mathcal{T}_{new}$. In fact, this analysis is a generalized formulation of Amdahl's style law for each computation model and scenario.

Acknowledgments

We are would like to thank Prof. J. A. Makowsky for valuable discussions as well as for his reading of the contribution and his many suggestions.

Finally we would like to thank the referees for their careful reading and constructive suggestions.

References

[1] J. Makowsky and E. Ravve, "Incremental model checking for decomposable structures," in *Mathematical Foundations of Computer Science (MFCS'95)*, ser. Lecture Notes in Computer Science, vol. 969. Springer Verlag, 1995, pp. 540–551.

[2] E. Ravve and Z. Volkovich, "A systematic approach to computations on decomposable graphs," 2013, to appear in Proceedings of SYNASC13.

[3] E. Ravve, Z. Volkovich, and G.-W. Weber, "Effective optimization with weighted automata on decomposable trees," 2013, optimization Journal Special Issue at ECCO 2013.

[4] S. Feferman and R. Vaught, "The first order properties of products of algebraic systems," *Fundamenta Mathematicae*, vol. 47, pp. 57–103, 1959.

[5] J. Makowsky, "Algorithmic uses of the Feferman-Vaught theorem," *Annals of Pure and Applied Logic*, vol. 126, pp. 159–213, 2004.

[6] L. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33(B), pp. 103–111, 1990.

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: towards a realistic model of parallel computation," in *PPOPP '93 Proceedings of the fourth ACM SIGPLAN Symposium on Principles and practice of parallel programming*, vol. 28(7), 1993, pp. 1–12.

[8] (2013, Jan) Seville map. [Online]. Available: http://www.lonelyplanet.com/maps/europe/spain/andalucia/seville/

[9] H. Ebbinghaus, J. Flum, and W. Thomas, *Mathematical Logic, 2nd edition*, ser. Undergraduate Texts in Mathematics. Springer-Verlag, 1994.

[10] J. Makowsky, "Translations, interpretations and reductions," 1994, unpublished Manuscript.

[11] R. Fagin, "Generalized first-order spectra and polynomial time recognizable sets," in *Complexity of Computation*, ser. American Mathematical Society Proc, R. Karp, Ed., vol. 7. Society for Industrial and Applied Mathematics, 1974, pp. 27–41.

[12] M. Garey and D. Johnson, *Computers and Intractability*, ser. Mathematical Series. W.H. Freeman and Company, 1979.

[13] J. Makowsky and Y. Pnueli, "Arity vs. alternation in second order definability," in *LFCS'94*, ser. Lecture Notes in Computer Science, vol. 813. Springer, 1994, pp. 240–252, (Extended version to appear in the Annals of Pure and Applied Logic, 1995).

[14] M. Vardi, "The complexity of relational query languages," in *STOC'82*. ACM, 1982, pp. 137–146.

[15] E. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier Science Publishers, 1990, vol. 2, ch. 16.

[16] J. Makowsky and E. Ravve, "Incremental model checking for fixed point properties of decomposable structures," 1995, technical Report TR844, revised version, April 1995, Department of Computer Science, Technion–Israel Institute of Technology, Haifa, Israel.

[17] N. Biggs, R. Damerell, and D. Sands, "Recursive families of graphs," *Journal of Combinatorial Theory*, vol. 12, pp. 123–131, 1972.

[18] B. Courcelle and J. Makowsky, "Fusion in relational structures and the verification of monadic second-order properties," *Mathematical. Structures in Comp. Sci.*, vol. 12, pp. 203–235, April 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=966880.966886

[19] E. Fischer and J. Makowsky, "Linear recurrence relations for graph polynomials," in *Pillars of computer science*, A. Avron, N. Dershowitz, and A. Rabinovich, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 266–279. [Online]. Available: http://dl.acm.org/citation.cfm?id=1805839.1805854

[20] D. Babić, A. Graovac, B. Mohar, and T. Pisanski, "The matching polynomial of a polygraph," *Discrete Applied Mathematics*, vol. 15, pp. 11–24, 1986.

[21] E. Cockayne, E. Hare, S. Hedetniemi, and T. Wimer, "Bounds for the domination number of grid graphs," *Congr. Numer*, vol. 47, pp. 217–228, 1985.

[22] D. Crandall. (2013, Jan) CSE 477 design project specifications report for the discrete cosine transform decoder. [Online]. Available: http://www.cs.indiana.edu/ djcran/projects/cse477/report3/report3-2.html

[23] A. Glikson and J. Makowsky, "NCE graph grammars and clique-width," in *WG'03*, 2003, pp. 237–248.

[24] G. Gottlob, E. Grädel, and H. Veith, "Datalog LITE: a deductive query language with linear time model checking," *ACM Transactions on Computational Logic*, vol. 3(1), 2002.

[25] E. Ravve, "Decomposition of databases with translation schemes," Ph.D. dissertation, Department of Computer Science, Technion–Israel Institute of Technology, Haifa, 1998.

[26] X. Quan and G. Wiederhold, "Incremental recomputation of active relational expressions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, no. 3, pp. 337–341, 1991.

[27] G. Dong and R. Topor, "Incremental evaluation of Datalog queries," in *Database Theory, 4th ICDT'92*, ser. Lecture Notes in Computer Science, J. Biskup and R. Hull, Eds., vol. 646. Springer Verlag, 1992, pp. 282–296.

[28] R. Neal and G. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants," in *Learning in Graphical Models*. Kluwer Academic Publishers, 1998, pp. 355–368.

[29] S. Ahn, J. A. Fessler, D. Blatt, and A. Hero, "Convergent incremental optimization transfer algorithms: Application to tomography," *IEEE Trans. Med. Imaging*, vol. 25(3), pp. 283–296, 2006.

[30] O. Şeref, R. Ahuja, and J. Orlin, "Incremental network optimization: Theory and algorithms," *Operations Research*, vol. 57, pp. 586–594, 2009.

[31] R. Poli, "Evolution of graph-like programs with parallel distributed genetic programming," in *Proceedings of 7th ICGA*, 1997, pp. 346–353.

 ISBN: 978-1-61208-346-9