UNIVERSITY OF ATHENS

ALGORITHMS, LOGIC AND DISCRETE MATHEMATICS

Master's Thesis

The Dynamic Complexity of Reachability and Related Problems

Stavros Taxiarchis Potsakis

Thesis committee:
Stathis Zachos
Ioannis Kokkinis
Archontia Giannopoulou
Dimitrios Thilikos

Athens, July 2021

Abstract

The reachability query is of particular interest for investigating the expressive power of DynFO, since it is one the simplest queries that cannot be expressed statically in first-order logic, but rather requires recursion.

We present the framework of Dynamic Complexity using notation from descriptive complexity. We study the class DynFO by showing which problems are in DynFO and the relation between DynFO and other classes such as P, P-complete, L-complete, NL-complete and DynAC^0. We also give the definition of "bounded-expansion reductions" from [7] which honor dynamic complexity classes.

Then we present a detailed proof of Reachability being in DynFO from [7] by using computational linear algebra problems. Futhermore, using the previous result and techniques from its proof, we give the proof of 2-Sat is DynFO and PerfectMatching and MaxMatching are in non-uniform DynFO from [7].

Contents

Chapter 1

Introduction

In traditionally computational complexity, we deal with static problems. More precisely, a static problem is a problem that for an input ,$\mathcal{A}$, we check whether this input satisfies a boolean query, Q. Static problems are very important for computational theory, but some problems are presented to us in a non-static form. A lot of problems can be described as dynamic, i.e. as problems that each time the input data are modified, i.e. objects are inserted or deleted. For example, when a person waits for the next train to come, but the train is cancelled on short notice, then the person has to find another train or a combination of trains so he will reach his destination as fast as possible.

Another example is that of GPS. When the GPS is always updating the route from one place to another, because of traffic or construction on some roads or accidents that stop the circulation. Another example is the compilation of programs. Staged compilation is a compilation strategy in which the code-compilation process is completed in multiple stages: for example, at static compile time, at link time, at load time, and at run time. But if we work with a large program and each time we want to change a part of it and then recompile, we need a way not to read and compile all the program again.

It is not always easy to compute a query after a change of data, because sometimes this leads to an amount of data that it is questionable if the efficiency or latency(lack of it) of them is guaranteed. That's why instead of static algorithms, we work with incremental algorithms. An incremental algorithm is given a sequence of input, and gives a sequence of solutions that build incrementally while adapting to the changes of the input. More specifically, incremental algorithms use auxiliary data to answer queries after a change on the input data.

In this thesis, we do not study incremental algorithms, but the complexity of problems that change dynamically. When we work in database, we deal with a sequence of small updates and queries to a large amount of data and the most important is that all of this work should be performed as fast as possible compared to the size of the database. So databases are dynamic. At first, it was necessary to find a way to describe how to maintain each query after an insertion or deletion and that language would give enough information about the time and space that was needed. Unfortunately, for many decades there were not appropriate Database Complexity Classes for database systems and traditional complexity classes are not very useful for database systems. But in 1995, Patnaik and Immerman published a paper which introduced complexity classes appropriate for database systems, more specifically dynamic complexity classes [24]. For that reason, we study the complexity of dynamic problems (or database problems) in the frame of descriptive complexity.

In 1974, Ron Fagin proved that the problems that belong to the class NP(non-deterministic polynomial time class) can be described by second order existential logic. In the next years, it has been shown that most of the traditional complexity classes have also a descriptive characterization [19]. Another example of a well-known complexity class, that has a descriptive characterization, is the deterministic polynomial time, or P, which is equal to FO(LFP) - first-order logic closed under the inductive definitions [19]. Also, it holds that the notions of naturalness in traditional complexity theory corresponds easily to the notion of naturalness in logic. Thus, in descriptive complexity, we can define any notion that was defined in traditional complexity theory. Indeed, any input can be presented as relational database which is a finite logical structure, with schemata (or vocabularies) that consist of the domain, relation symbols and constant symbols. Also we define a set of structures that depend on a specific schema. Since each complexity problem is set of of inputs which satisfy a common property then in descriptive complexity we define them as subset of a set of structures for some schema.

One of the most commercial query languages for the databases systems is SQL. We use SQL as query language because it is more convenient to deal with it than first-order logic. Also, in database theory, an important result is that SQL is relational "complete". i.e. it can express all relational queries expressible in calculus. Relational calculus is equivalent to first-order logic [1], and that's why first-order logic is a very interesting language in database theory and very useful to express our efforts of maintaining queries after insertion or deletions. Thus, first-order logic is very powerful as an update mechanism and is in the center of research for many decades. We already know that first-order logic with arithmetic(more specifically relations for $<$, $+$, $\times$) is as powerful as uniform AC^0 circuits, and so AC^0 is the complexity class that consists of the problems that is the most likely to belong to the dynamic complexity class of first-order logic. Therefore, we study dynamic problems, that can be maintained by the minimum possible number of resources.

In the last paragraph, we mention the term dynamic complexity class of first-order logic and this class is denoted as DynFO. More specifically, the complexity class DynFO consists of all the queries that can be maintained after single tuple insertions and deletions using first-order formulas (or AC^0-circuits). Many queries can be maintained by first-order logic. The reachability query is a query that have occupied researchers for many years and can be defined as follows: for a given graph G, it returns all pairs (s, t) of vertices, for which there is a path from s to t. The reachability query represent the transitive closure of a relation and also is a well known NL-complete [19], and so it was one of the problems that researchers tried to put it in DynFO for many years. Also, it is one of the most basic problems in graph theory and also is the most important problem in computer science, since represents the configuration graph of a computation from the initial state to the final state (accepting or rejecting).

Many version of the reachability query have been under investigation for membership in DynFO. Immerman and Patnaik have shown that reachabiltiy in undirected graphs, on deterministic graphs and in acyclic graphs are in DynFO for several decades now [24]. "Reachability in directed graph is in DynFO" is a question that troubled researchers for many years and Immerman and Patnaik had it as an open problem. Through the years, reachability in directed graphs has been studied for several variants of DynFO. William Hesse proved in 2001 that the dynamic complexity of transitive closure (or the reachability query) is in DynTC^0, i.e. he proved that the reachabiliy quey can be maintained by quantifier-free updated formulas [15]. Eventually, around 2015 came the proof that reachability in directed graphs is in DynFO [7].

Finally, we show that two queries that deal with matchings in graphs can be maintained in a non-uniform extension of DnyFO. More specifically, we will give the proof of that PerfectMatching and MaxMatching are in non-uniform DynFO from [7]. Here, PerfectMatching is the Boolean query that asks for the existence of a perfect matching in a graph, whereas MaxMatching returns the size of the maximum matching. Non-uniform DynFO is the extension of DynFO whose programs can use arbitrarily pre-defined auxiliary relations.

In Chapter 2, we fix our notation for database, structures and queries. Based on [7], we give the definition of the dynamic complexity framework and give some problems in DynFO from [7]. We give the definition of the first-order reductions, bfo reductions and the bfo-tt reductions from [7] [19]. Later, we give the proof of some results between DynFO and other complexity classes from [19]. In chapter 3, we fix our notation for linear algebra. Chapter 4 is dedicated to the proof that the reachability query is in DynFO [7]. Chapter 5 presents the results on 2-Sat and graph matchings. The conclusion and the further work is given in Chapter 6.

Chapter 2

Dynamic Complexity

2.1 Descriptive Complexity

As we have already seen, in descriptive complexity, an input can be represented as a relational database and relational databases are considered as logical structures. Thus, for all the notions we will need to define so that we give the descriptive characterization of problems and study their complexity, we will use the notation from relational database. More specific, we define the relational schema, the structure over a schema, the query and we will talk about the arithmetic for d-tuples, d$\geq$1.

2.1.1 Schema and database

A schema(or a vocabulary) is a tuple of relation symbols, function symbols and constant symbols. All relation symbols R_i are characterized by a number, a_i, which denotes their arity. Arity is the number of argument taken by a relation. Similarly, function symbols have an arity. A well-known example of schema is

$$\tau_F = \langle 0, 1, +^2, \times^2 \rangle,$$

the schema of a field, which consists of two constant symbols 0 and 1, and two functions symbols of arity 2.

If a schema does not contain function symbols then we call it relational schema. More formally, a relational schema τ(or a relatioanl vocabulary) consists of a set τ of relation symbols, accompanied by an arity function Ar: $\tau_{rel} \to \mathbb{N}$, and a set τ_{const} of constant symbols. An example of relational schema is $\tau_g = \langle E^2$, s, t$\rangle$, the relational schema of graphs with two specific endpoints, which consists of a relation symbol E with Ar(E)=2 (the edge relation) and two constant symbols s and t that correspond to the two endpoints. Another example is $\sigma = \langle \leq^2, \mathrm{S}' \rangle$, the relational schmea of binary strings, which consists of two relation symbols $\leq$ with Ar($\leq$)=2 (or the ordering relation) and S with Ar(S)=1.

A relational database (or a database) is a finite logical structure. A database $\mathcal{D}$ over schema τ consists of a domain D, a set of relations and a set of constants. More precisely, a database $\mathcal{D}$ with domain D assign to every relation symbol R $\in \tau_{rel}$ a relation of arity Ar(R) over D and to every constant symbol c $\in \tau_{const}$ an element from D. Also, except from domain, we define the active domain and will be denoted as adom(D), which contains of all the elements of domain D that occur in some relation or as a constant.

An example of a relational database is a genealogical database

$$\mathcal{A}_0 = \langle U, F^1, P^2, S^2 \rangle,$$

where U is the domain of the database, F is a relation symbol that has been assign to a relation of arity Ar(F)=1 and P, S is relation symbols that have been assigned to relations of arity Ar(P)=Ar(S)=2. Moreover, F is true for the female elements of U, P and S is true for the parents and spouses, respectively.

2.1.2 Structures and arithmetic

A structure $\mathcal{S}$ with relational schema τ is a tuple

$$\langle D, R_1^{\mathcal{S}}, ..., R_r^{\mathcal{S}}, c_1^{\mathcal{S}}, .., c_s^{\mathcal{S}} \rangle.$$

More specific, is a pair (D,$\mathcal{D}$), where D (or dom($\mathcal{S}$)) is a domain and $\mathcal{D}$ is a database with domain over schema τ. The domain D is a finite and nonempty set. For every relation symbol R with Ar(R) = k in τ, $\mathcal{S}$ has a relation $R^{\mathcal{S}}$ with arity k defined on the domain D. For every constant symbol c in τ, $\mathcal{S}$ has a constant $c^{\mathcal{S}}$ which corresponds to a particular element of the domain D. For example, a structure of the relational schema of graphs τ_g is the tuple

$$\langle \{0, 1, ..., n\}, E^{\mathcal{S}}, s^{\mathcal{S}}, t^{\mathcal{S}} \rangle$$

where the domain corresponds to the vertices of the graph.
The structure

$$\langle \{0, 1, ..., 6\}, \{(0, 3), (1, 2), (1, 4), (2, 6), (3, 5), (5, 6)\}, 1, 4 \rangle$$

of schema τ_g defines the graph with 7 vertices, 6 edges and two specific vertices 1 and 4.

An another example is a structure of the relational schema σ of a binary string, which is the tuple

$$\langle\{0, 1, ..., n\}, \leq^{\mathcal{S}}, S^{\mathcal{S}}\rangle$$

where the elements of the domain correspond to bits of that string. The structure

$$\langle\{0, 1, ..., 6\}, \leq^{\mathcal{S}}, \{2, 3, 4\}\rangle$$

of schema σ defines the binary string that is one in the positions 2, 3, 4. Also, a structure of the schema λ=$\langle$F^1, S^2, P$^2\rangle$ is a tuple

$$\langle U, F^{\mathcal{S}}, S^{\mathcal{S}}, P^{\mathcal{S}}\rangle$$

where F is unary relation which is true for all the female elements of U, P and S is binary relations for parent and spouse, respectively. The structure

$$\mathcal{S} = \langle\{a, b, c, d, o, t\}, \{c, d, o\}, \{(a, c), (b, o)\}, \{(a, b), (c, b), (b, t), (b, d), (o, d), (o, t)\}\rangle$$

of the schema λ defines the genealogical tree of the set of people {a, b, c, d, o, t}. The people c, d, o are female, the people a,c and b,o are spouses and a,c are parents of b and b,o are parents of o and t.

In descriptive complexity, a problem is a set of inputs and an input is a logical structure, so a problem will be a subset of a set that consists of all structures for a specific schema, and we will denote as STRUCT[τ] for a schema τ. Also, a problem is the same thing as a boolean query.

In the previous paragraphs, we give the definitions of databases and structures. All of the readers may be confused why this two notions are different. The difference between structure and database is that the domain of a structure is static but the domain of the database, or more specific the active domain, may change if we insert or delete a tuple from a relation or set a new constant.

Many times the domain of a logical structure is not in order, i.e. between the elements of the domain there is no ordering that will say an element is lesser or grater from another element. But when we discuss about computation, it is necessary for the domain to be ordered. A domain is in order if the schema of its structure contains the binary arithmetic relation $\leq$ (or linear order). When the schema of a structure has $\leq$ as its element, then this structure must always consider that relation as the total ordering of its domain. Also, when linear order is contained in a schema τ, then the domain D of every τ-structure has 1-1 correspondence with the set {0, 1,..., |D|-1}. Moreover, a schema may contain more arithmetic relations like +

and $\times$, which are interpreted by the relation of addition and multiplication on the domain elements, respectively. Also, we define $\mathcal{L}(\tau)$ be the first-order language built up from the elements of the schema τ and the arithmetic relations, $=$, $\leq$ and BIT ,and arithmetic constants min, max using logical connectives: $\vee$, $\wedge$, $\neg$, variables: x, y, z, ..., and quantifiers: $\forall$, $\exists$. The constants min and max correspond to the minimum and maximum element of the domain in the ordering $\leq$ and BIT(x,y) is true if and only if the y-th bit of the binary representation of x is one.

There may be times that a schema will need to use arithmetic for d-tuples, where d $\geq$1. If d=1, then it is the arithmetic we are already known. But, when d $>$ 1, then the arithmetic is defined differently. Now we will see how the arithmetic lifts for d=2 in first-order logic and for arbitrary d the proof exists in [29].

Theorem 2.1.1. *The arithmetic can be lifted to relations over 2-tuples in first-order logic.*

Proof. A tuple $\vec{x} = (x_0,x_1) \in \underline{N}^2$, where $\underline{N} = \{0,1,....,N\}$, corresponds to the number $x_1 + (1+N)x_0$.

The relation $<$ for comparing a 2-tuple with another 2-tuple can be defined in first-order logic as follows:

$$\varphi^2_<(x_0,x_1,y_0,y_1) := x_1 < y_1 \vee (x_1 = y_1 \wedge x_0 < y_0)$$

and the relation $\leq$ is the reflexive closure of $<$.

The relation $+$ for 2-tuples can be defined in first-order logic as follows. For 1-tuples we have that $\varphi^1_+(x_0,y_0,z_0,z_1)$ expresses that $x_0 + y_0 = z_1(N+1) + z_0$. If $x_0 + y_0 \leq N$, then $x_0 + y_0 = z_1(N+1) + z_0$ if and only if $z_1 = 0$ and $z_0 = x_0 + y_0$. Otherwise, we have $N+1 \leq x_0 + y_0 \leq 2(N+1)$, and hence there are u, v $\in$ N such that $x_0 + u = N$ and $u + v = y_0 - 1$. Thus $x_0 + y_0 = x_0 + u + 1 + v = (N+1) + v$. Hence, $x_0 + y_0 = z_1(N+1) + z_0$ if and only if $z_1 = 1$ and $z_0 = v$.

For the formula of the relation $+$ for 2-tuples, we have that $\varphi^2_+(x_0,x_1,y_0,y_1,z_0,z_1,z_2,z_3)$ expresses that $x_0 + y_0 = z_1(N+1) + z_0$ and $x_1 + y_1 + z_1 = z_3(N+1) + z_2$ in similar way as above. The formula φ^2_+ can be easily defined by using formula φ^1_+.

The relation $\times$ for 2-tuples can be defined in first-order logic as follows. To expose the overall idea, we consider the multiplication of two decimal numbers. For example,

$$540 \times 720 = (500 + 40) \times (700 + 20) =$$

$$(500 \times 700) + (500 \times 20) + (40 \times 700) + (40 \times 20)$$

and with the same way we will give the definition of multiplication for 2-tuples from [29]. For $\vec{x} = (x_0, x_1)$ and $\vec{y} = (y_0, y_1)$ we have that:

$$\varphi^2_\times(\vec{x}, \vec{y}) := \varphi^1_+(\varphi^1_\times(x_1, y_1), \varphi^1_+(\varphi^1_\times(x_1, y_0), \varphi^1_+(\varphi^1_\times(x_0, y_1), \varphi^1_\times(x_0, y_0))))),$$

i.e., $(x_0, x_1) \times (y_0, y_1) = [(x_1 \times y_1) + [(x_0 \times y_1) + [(x_1 \times y_0) + (x_0 \times y_0)]]]$. $\square$

2.1.3 Queries

In the previous sections, we mention the concept of the query as an example of computation. A query I is mapping from structures of one schema to structures of another schema. More precisely, if it is given us a problem over the schema τ or a subset of STRUCT[τ] then a query maps this subset to another subset of STRUCT[σ] or a problem over the schema σ. The query I is always polynomial bounded, i.e. there is a polynomial p such that for every structure $\mathcal{A}$ $\in$STRUCT[τ], $||I(\mathcal{A})|| \leq p(||\mathcal{A}||)$. For example, the query I_{add}, which for a given pair of binary strings A and B, returns their addition. Query I_{ab} is the mapping from STRUCT[τ_{ab}] to STRUCT[τ_s]. The τ_{ab} is the schema that consists of the domain D, where ||D||=n, the total ordering, and two unary relations A and B that A(i) is true if and only if the i bit of A is one and similarly for B and the τ_s is the schmema $\langle \leq^2, S^1 \rangle$ that defines binary strings. It is again obvious that this query is polynomial bounded, since $||I(\mathcal{A})|| = ||\mathcal{A}||$.

A boolean query I is mapping from structures of a schema σ to the set {0,1}. For example, the query DIAM[8] on graphs is true if and only if the diameter of the graph is at most 8. This query is the mapping from STRUCT[τ_g] to {0,1}, where τ_g is the schema of graphs and it is obvious that query DIAM[8] is polynomial bounded.

2.2 Dynamic Complexity

2.2.1 Dynamic Complexity Classes

We have already seen that in dynamic complexity, inputs are represented as relational structures and the domain is fixed from the beginning, but the database in the initial structure is empty and is then modified by a sequence of insertions and deletions of tuples. A change operation is thus of the form **insert $\vec{t}$ into R or delete $\vec{t}$ from R**, for some tuple $\vec{t}$ and input relation R or **set constant c_j to a**, where a is element of the domain. More precisely, we denote by $\mathcal{R}_{n,\sigma}$ the set of all possible change operations, where n denotes the size of the domain of the structure with schema σ.Let α be a sequence of change operations and $\mathcal{I}$ an input database, then $\alpha(\mathcal{I})$ denotes the structure we get after applying α to $\mathcal{I}$.

Let $\mathcal{I}_0$ be the initial structure and S$\subseteq$STRUCT[σ] be a boolean query. After the first insertion or deletion of a tuple there is the query as to if $\mathcal{I}_1$ satisfies the problem S, where $\mathcal{I}_1$ is the result structure of applying that change operation to $\mathcal{I}_0$. In order to prove that $\mathcal{I}_1$ satisfies S, we maintain an auxiliary structure $\mathcal{A}_1$. After the i-th change operation, i.e. after that the i-th insertion or deletion modifies the current structure, we want to maintain the corresponding auxiliary structure $\mathcal{A}_i$, so that to check if the input structure $\mathcal{I}_i$ satisfies S. Let $\mathcal{A}_0$,$\mathcal{A}_1$,...,$\mathcal{A}_r$ be the sequence of auxiliary structure, the sequence a=a_1,...,a_r of change operations. We compute $\mathcal{A}_0$ from the corresponding $\mathcal{I}_0$ and $\mathcal{A}_i$ can compute efficiently from $\mathcal{A}_{i-1}$ and change operation a_i. Each $\mathcal{A}_i$ is not polynomial larger than the $\mathcal{I}_i$ and if $\mathcal{A}_i$ is given then we can test whether $\mathcal{I}_i$ satisfies S. Now we give the formal definition for dynamic complexity classes.

Definition 2.2.1. *Let $\mathcal{C}$ be a a static complexity class. Let σ= $\langle R_1^{a_1},R_2^{a_2},\ldots,R_t^{a_t},c_1,\ldots,c_s\rangle$ be a schema and let $S \subseteq$ STRUCT[σ] be any problem. Let $\mathcal{I}_0$,$\mathcal{I}_1$,...,$\mathcal{I}_r$ be the sequence as it defined above. Let $n \in \mathbb{N}$ and let a_1,...,a_r be any sequence of any change operation from $\mathcal{R}_{n,\tau}$.*

Dynamic complexity class Dyn$\mathcal{C}$ is the set of boolean queries $S \subseteq$ STRUCT[σ] that satisfy the following conditions:

- *There exists another boolean query, $T \subseteq$ STRUCT[τ] in static $\mathcal{C}$ that serves as auxiliary structure for the dynamic algorithm.*
- *There exists mappings f : $\mathcal{R}^*_{n,\sigma}$[1] $\rightarrow$ STRUCT[τ] and g : STRUCT[τ]$\times\mathcal{R}_{n,\sigma}$ $\rightarrow$ STRUCT[τ] and a constant $k \in \mathbb{N}$ such we have the following.*
  - ■ *The maps f and g determine a sequence $\mathcal{A}_0$,$\mathcal{A}_1$,...,$\mathcal{A}_r$ $\in$ STRUCT[τ] such that for all $i \leq r$,*

    (i) $\mathcal{A}_0$ = *f(n)*

    (ii) $||\mathcal{A}_i|| \leq n^k$

[1] $\mathcal{R}^*_{n,\sigma}$ is the set of finite sequence from $\mathcal{R}_{n,\sigma}$.

(iii) $\mathcal{A}_i = g(\mathcal{A}_{i-1}, a_i)$, *and*
(iv) $\mathcal{I}_i \in S \Leftrightarrow \mathcal{A}_i \in T$

Thus, we can efficiently, i.e., in static $\mathcal{C}$ for each change operation and query, maintain the auxiliary structure $\mathcal{A}_i$, and test whether $\mathcal{I}_i \in S$ by testing whether $\mathcal{A}_i \in T$.

Let $\mathcal{P}$ denote the dynamic algorithm that works on as input structure $\mathcal{I}$ and maintains an auxiliary structure $\mathcal{A}$ as we see in Definition 2.2.1.. Also, a dynamic program $\mathcal{P}$ has a set of maintaining rules that specify how auxiliary relations are maintained after a change operation. For a change operation δ, we denote the maintaining state for input structure $\mathcal{I}$ and auxiliary structure $\mathcal{A}$ by $\mathcal{P}_\delta(\mathcal{I},\mathcal{A})$ and for a change sequence α, we denote the maintaining state by $\mathcal{P}_\alpha(\mathcal{I},\mathcal{A})$.

The classes that we will study are:

- DynFO: the class of all dynamic queries that can be maintained by a dynamic algorithm with formulas from first-order logic starting from an empty input and auxiliary structures.
- DynFO(+, ×): the same as DynFO, except that the algorithm have three particular auxiliary relations that are initialized as a linear order and the corresponding addition and multiplication relations.
- DynAC^0: the class of all dynamic queries that can be maintained by uniform AC^0-circuits,

In the next section, we will give the proof of that DynFO(+, ×) = DynAC^0 from [12].

2.2.2 DynFO(+,×) = DynAC^0

It is well known that first-order logic with arithmetic is as powerful as uniform AC^0-circuits [3]. This correspondence naturally transfers to the dynamic setting. That is, a query can be maintained in DynFo(+,×) if and only if it can be maintained by uniform AC^0 circuits.

Let the activated domain (ad) of the structure be those elements of a k-tuple that have at some point during this sequence of change operations inserted or deleted from a relation. The differences between the domain, elements of the active domain, and the activated domain are that the domain contains all the elements that can be used in relations and does not change during a dynamic computation, the active domain adom($\mathcal{D}$) of a database $\mathcal{D}$ consists of all elements that occur in some tuple of $\mathcal{D}$ after a change sequence a and the activated doamin consists of all elements that occur somewhere in a. For example, adding the edges (1,2) and (2,3) to an initially empty graph over the domain {1,2,3,4} and then deleting edge {1,2}, it gives us an input databse with active domain {2,3} and activated elements 1,2,3. By that example, we conclude that every elements in the active domain is activated but not vice versa.

Let ad(x) be the predicate that defines this set. Let $\leq_{\mathsf{ad}}$(x,y) denote a total ordering on the activated domain, defines by the order which elements entered the activated domain. To resolve ties, when INSERT(G,a,b) occurs, a $\neq$ b and a, b $\notin \mathsf{ad}$, we arbitrarily choose to let element a into the activated domain prior to element b. Once we have an ordering $\leq_{\mathsf{ad}}$, we can view ad as a set of numbers {1,...,n$_{\mathsf{ad}}$}. Let max$_{\mathsf{ad}}$(x) be the predicate that hold only for current maximum, n$_{\mathsf{ad}}$. Let BIT$_{\mathsf{ad}}$(x,i) denote the bit predicate on these numbers, meaning " the i-th bit of the number x is on"(we assume bit position are numbered starting at 1). Let $+_{\mathsf{ad}}$(x,y,z) and $\times_{\mathsf{ad}}$(x,y,z) be graphs of the ordinary addition and multiplication function on the domain ad.

Lemma 2.2.2. *We can built and maintain auxiliary relations that defines ad(x), $\leq_{\mathsf{ad}}$(x,y), max$_{\mathsf{ad}}$(x), BIT$_{\mathsf{ad}}$(x,i), $+_{\mathsf{ad}}$(x,y,z) and $\times_{\mathsf{ad}}$(x,y,z).*

Proof. All these relations do not change after a delete operation, so we only write formulas for updates after insertion.

$$k_{\mathsf{ad}}^{\mathsf{insert}}(x,a,b) \equiv \mathsf{ad}_{\mathsf{old}}(x) \vee x = a \vee x = b$$

$$\begin{aligned} k_{\leq_{\mathsf{ad}}}^{\mathsf{insert}}(x,y,a,b) \equiv\; & \leq_{\mathsf{ad}_{\mathsf{old}}}(x,y) \vee (k_{\mathsf{ad}}^{\mathsf{insert}}(y,a,b) \wedge \neg \mathsf{ad}_{\mathsf{old}}(y) \wedge \\ & (x = y \vee \mathsf{ad}_{\mathsf{old}}(x))) \vee \\ & (k_{\mathsf{ad}}^{\mathsf{insert}}(x,a,b) \wedge k_{\mathsf{ad}}^{\mathsf{insert}}(y,a,b) \wedge \\ & \neg \mathsf{ad}_{\mathsf{old}}(x) \wedge \neg \mathsf{ad}_{\mathsf{old}}(y) \wedge (x = a \wedge (y = a \vee y = b)) \end{aligned}$$

$$k_{\mathsf{max}_{\mathsf{ad}}}^{\mathsf{insert}}(x,a,b) \equiv k_{\mathsf{ad}}^{\mathsf{insert}}(x,a,b) \wedge \forall z (k_{\mathsf{ad}}^{\mathsf{insert}}(z,a,b) \rightarrow k_{\leq_{\mathsf{ad}}}^{\mathsf{insert}}(z,x,a,b))$$

To express BIT$_{\mathsf{ad}}$(x,i), observe that the only interesting situation is when x is new to ad. Using the fact that addition is first-order expressible, we can identify the prior maximum element, z, of ad and increment its binary representation by 1 or 2, depending on whether x is first or second new entry into ad.

$$\begin{aligned}
k^{\mathsf{insert}}_{\mathsf{BIT}_{\mathsf{ad}}}(x, i, a, b) \equiv \mathsf{BIT}_{\mathsf{ad}_{\mathsf{old}}}(x, i) \vee \\
((\neg \mathsf{ad}_{\mathsf{old}}(x) \wedge (x = a \vee (\mathsf{ad}_{\mathsf{old}}(a) \wedge x = b)) \wedge \\
\exists z(max_{\mathsf{old}}(z) \wedge \exists y(\mathsf{ad}(y) \wedge \neg \mathsf{BIT}_{\mathsf{old}}(z, y) \wedge \\
\forall j(j <_{\mathsf{ad}} y \to \mathsf{BIT}_{\mathsf{old}}(z, j)) \wedge \\
((i > y \wedge \mathsf{BIT}_{\mathsf{old}}(z, i))) \vee i = y))) \vee \\
\text{(increment the binary representation by 2, similarly)}
\end{aligned}$$

the maintainability of $+_{\mathsf{ad}}$(x,y,z) and $\times_{\mathsf{ad}}$(x,y,z) follows from the known fact that + and $\times$ are first-order expressible in terms of $\leq$ and BIT. □

Lemma 2.2.2. sheds some light on why lower bound proofs for Dyn-FO are so hard. Let Dyn-(uniform)-AC^0 be the class of dynamic problems where it is possible to update both the query and the auxiliary structure with a Dlogtime-uniform constant-depth, polynomial-size boolean circuits with unbounded fan-in, where the amount of auxiliary structure is polynomial long in the length of the input string, and the length of the input string is fixed.

Let Dyn-FO$^\$$ be the same as Dyn-FO except that we assume the entire universe is activated. This could be enforced by, e.g., assuming that starting from the $\emptyset$ structure, a k-tuple is inserted and deleted from the relation.

Corollary. Dyn-FO$^\$$ = Dyn-AC^0.

Proof. It was shown that uniform-AC^0 is equivalent to FO($\leq$,BIT), first-order logic equipped with the given built-in arithmetic predicated. Once the entire universe is activated, by above Lemma we can build all the necessary arithmetic predicates on the universe dynamically. □

2.2.3 Problems in DynFO

Consider the simple boolean query: Parity, which is true if and only if the input binary string has an odd number of one's. We know that Parity is not in static FO [14] [2]. The dynamic algorithm for Parity maintains a bit b which is switched after any change to the string. Moreover, it is very important to remember the input string so that we can tell if a request has actually changed the string.

Example 2.2.3. *The schema of the Parity problem is τ that consists of the relation M with Ar(M)=1 and zero constants. Let $\mathcal{I}$ be the input structure over schema τ coding the binary string w. Then $\mathcal{I}_w \models M(i)$ iff the i-th bit of w is a one. Let σ be a schema that consists of the relation M with Ar(M)=1 and one constant b where b is a boolean constant symbol. Let $\mathcal{A}$ be an auxiliary structure over schema σ.*

The initial auxiliary structure $\mathcal{A}_\emptyset = \langle\{0,1,\ldots n\text{-}1\},\emptyset,false\rangle$ consists of a string of all 0's, with the boolean b initialized to false and from Definition 2.2.1. we have that $f(n) = \mathcal{A}_\emptyset$.

To show that Parity is in DynFO we need to give first-order formulas that maintaining the auxiliary relations after a change operation. The cases of change operations are the setting of a bit of w to 0 or 1. Let M,b denote the relations in the auxiliary structure $\mathcal{A}$ before the change, and M',b' are their values afterwards which have produced by the function g. Depending on whether the change operation is an insert or delete, the two cases are:

ins(a,M):

$$M' \equiv M(x) \vee x = a$$

$$b' \equiv (b \wedge M(a)) \vee (\neg b \wedge \neg M(a))$$

del(a,M):

$$M' \equiv M(x) \wedge x \neq a$$

$$b' \equiv (b \wedge \neg M(a)) \vee (\neg b \wedge M(a))$$

It is well known that the graph reachability problem in not first order expressible and this often been used as a justification for using database query languages more powerful than FO.

The reachability query is a query that has troubled researchers for many years. Much time and effort have been put into showing that reachability query belongs to DynFO. At first, Immerman and Patnaik prove that reachability query in undirected graphs, on deterministic paths and in acyclic graphs are in DynFO[1].

Now we will give the proof that the graph reachability problem Reach_u from [19], that is the restriction of Reach to undirected graphs is in DynFO.

Theorem 2.2.4. *Reach_u is in DynFO.*

Proof. The schema of the input structure consists of a single relation E with Ar(E)=2. The schema of the auxiliary structure for this problem consists of the relations F with Ar(2) and PV with Ar(PV)=3 and the input relation, E. F(x, y) means that the edge (x, y) is in the current spanning forest. PV(x, y, u) means that there is a (unique) path in the forest from x to y via vertex u. The vertex, u, may be one of the endpoints. So, for example, if F(x, y) is true, then so are PV(x, y, x) and PV(x, y, y). The goal is to maintain a spanning forest of the underlying graph via those relations. We maintain the undirected nature of the graph by interpreting insert(E, a, b) or delete(E, a, b) to do the operation on both (a, b) and (b, a).

Insert (E, a, b):

We denote the updated relations as E$'$ F$'$, and PV$'$. Also, we shall use P(x, y) to abbreviate (x=y $\vee$ PV(x, y, x)), and Eq(x, y, c, d) to abbreviate the formula,

$$((x = c \wedge y = d) \vee (x = d \wedge y = c)).$$

Maintaining the input edge relation is trivial:

$$E'(x, y) \equiv E(x, y) \vee Eq(x, y, a, b).$$

The edges in the forest remain unchanged if vertices a and b were already in the same connected component. Otherwise, the only new forest edge is (a, b).

$$F'(x, y) \equiv F(x, y) \vee (Eq(x, y, a, b) \wedge \neg P(a, b)).$$

Now all that remains is to compute PV$'$. The latter changes if and only if edge (a, b) connects two formerly disconnected trees. In this case, all new tuples (x, y, z) have x coming from one of the trees containing a and b, and y coming from the other:

$$PV'(x, y, z) \equiv PV(x, y, z) \vee (\exists uv)[Eq(u, v, a, b) \wedge P(x, u) \wedge P(v, y)$$

$$\wedge (PV(x, u, z) \vee PV(v, y, z))].$$

Delete (E, a, b):

If the edge (a,b) is deleted then E$'$(a,b) is set to false, but the relations F, PV change only if this edge was already in the forest, otherwise remain unchanged. When the edge is in the forest, then it is necessary to identify the vertices of the two trees in the forest created by deletion and choose (if any) another edge which by its insertion will connect those two trees and so updating the relations F, PV.

We define a temporary relation T to denote the PV relation after (a, b) is deleted, before the new edge, e, is inserted.

$$\mathsf{T}(x, y, z) \equiv \mathsf{PV}(x, y, z) \wedge \neg(\mathsf{PV}(x, y, a) \wedge \mathsf{PV}(x, y, b)).$$

Using T, we then pick the new edge that must be added to the spanning forest. New(x, y) is true if and only if edge (x, y) is the minimum edge that connects the two disconnected components:

$$\mathsf{N}ew(x, y) \equiv \mathsf{E}(x, y) \wedge \mathsf{T}(a, x, a) \wedge \mathsf{T}(b, y, b) \wedge$$

$$(\forall uv)[(\mathsf{E}(u, v) \wedge \mathsf{T}(a, u, a) \wedge \mathsf{T}(b, v, b)) \to (x < u \vee (x = u \wedge y \leq v))].$$

E$'$, F$'$, and PV$'$ are then defined as

$$\mathsf{E}'(x, y) \equiv \mathsf{E}(x, y) \wedge \neg \mathsf{Eq}(x, y, a, b).$$

We remove (a, b) from the forest and add the new edge:

$$\mathsf{F}'(x, y) \equiv (\mathsf{F}(x, y) \wedge \neg \mathsf{Eq}(x, y, a, b)) \vee \mathsf{N}ew(x, y) \vee \mathsf{N}ew(y, x).$$

The paths in the forest, from x to y via z, that did not pass through a and b, are valid. Also, new paths have to be added as a result of the insertion of a new edge in the forest:

$$\mathsf{PV}'(x, y, z) \equiv \mathsf{T}(x, y, z) \vee [(\exists u, v)(\mathsf{N}ew(u, v) \vee \mathsf{N}ew(v, u)) \wedge \mathsf{T}(x, u, x)$$

$$\wedge \mathsf{T}(y, v, y) \wedge (\mathsf{T}(x, u, z) \vee \mathsf{T}(y, v, z))].$$

□

We next give a new DynFO algorithm for the problems Reach$_d$ and Reach (acyclic). The problem Reach$_d$ is the restricted version of Reach in which we only allow deterministic paths, i.e. the edge (u,v) in on the path, then this must be the unique edge leaving u. By Reach(acyclic) we mean the Reach problem restricted to qieries in which the input graph is acyclic during its entire history.

Theorem 2.2.5. *Reach$_d$ and Reach(acyclic) are in DynFO.*

Proof. In the next sections we will give the proof that $\mathrm{Reach_d}$ is reducible to $\mathrm{Reach_u}$ from [19] and since $\mathrm{Reach_u}$ is in DynFO, $\mathrm{Reach_d}$ is in DynFO(that follow from Theorem ,which we will see later).

Now we will give an DynFo algorithm for Reach(acyclic). At first, we assume that every insertion will preserve acyclicity. For this problem the schema of the auxiliary structure consists of relation P(x,y) which means that there is a path from x to y in the graph. We maintain this relation as follows:

Insertion(E,(a,b)):[2]

$$P'(x,y) \equiv P(x,y) \vee (P(x,a) \wedge P(b,y))$$

Delete(E,(a,b)),:

$$P'(x,y) \equiv P(x,y) \wedge [\neg P(x,a) \vee \neg P(b,y) \vee (\exists uv)(P(x,u) \wedge P(u,a) \wedge E(u,v)$$

$$\wedge \neg P(v,a) \wedge P(v,y) \wedge (v \neq b \vee u \neq a))].$$

If there is a path from x to y passing through the edge (a, b), then P'(x,y) will always be false. Otherwise there is a path from x to y passing through the vertex u, where u is the last vertex in this path from which a is reachable. Note that $u \neq y$ because the graph was acyclic before the deletion of edge (a, b). Thus, the edge (u, v) described in the above formula must exist and acyclicity insures that the path $x \to u \to v \to y$ does not involve the edge (a, b). ☐

Theorem 2.2.6. *Minimum Spanning Tree can be computed in DynFo.*

Proof. For this problem we will maintain the forest edges and non-forest edges dynamically and the relations PV(x,y,e) and F(x,y) as in the case of $\mathrm{Reach_u}$. A big difference from $\mathrm{Reach_u}$ is that we have to maintain the minimum spanning tree and not a spanning tree. In this problem also exists a function $w : E \to \mathbb{N}$, which gives the weight of every edge.

Insert(E,(a,b)): When the edge (a,b) is inserted, there exist only two cases:

1) There is no path between a and b, and then (a,b) is in the forest , and PV is updated as in $\mathrm{Reach_u}$.

[2]E is the binary input relation.

2) There is path between a and b. We find all the edge that appear in this path and check if the weight of (a,b) is less from the weight of anyone of them. If not, then (a, b) cannot be a forest edge. Otherwise, if the (c,d) is the maximum weighted edge from the path, then we set F'(x,y) to false and F'(a,b) to true and update PV accordingly.

Delete(E,(a,b)): After the deletion of the edge (a,b) two trees have split. So we determine all vertices that are reachable from the vertices a and b, respectively. Then we order them by their weight and choosing the minimum and insert it. If there is more than one minimum, we break with tie with the lexicographic order[3]. At the end, PV is updated accordingly to reflect the merging of two disconnected trees into one. □

We conclude this section with another low-level problem that is in DynFO:

Proposition 2.2.7. *Multiplication in in DynFO*

Proof. Given two n-bit number, x, y, their addition can be expressed in FO⊆Dyn-FO. We maintain the product in a bit array, P. Suppose the change operation is Change(x,i,b), i.e. change the i-th bit of x to b. (Change(y,i,b) is analogous). There are two cases:

If the bit is changed from 0 to 1, then P' is given by shifting y to i bits to the left and then adding it to P. It is easily accomplished by a first-order formula. Indeed, we have the auxiliary relation auxY[j+i] which is true iff the j-th position of the binary representation of Y is one

$$\mathsf{auxY}[k] = (k > i) \wedge \mathsf{Y}[k - i]$$

and for every i

$$\mathsf{P}'[i] = \mathsf{P}[i] \oplus \mathsf{auxY}[i] \oplus \varphi_{\mathsf{carry}}(i)$$

If the bit is changed from 1 to 0, then P' is given by shifting y by i bits to the left, and adding the 2's complement of the resulting number with respect to $2^{|P|}$ to P[4]. Again this is easily accomplished by a first-order formula. Indeed, we define the auxiliary relation co_{N}Y which the 2's complement of Y with

[3]Here, lexicographic order means that we choose the first non-forest edge that reconnects the two disconnected trees.

[4]|P| is the length of the binary number P.

respect to 2^N and One[i] is true iff the i-th position of 1 is one, so for every i

$$\mathsf{co}_{|P|}Y[i] = (\neg auxY[i]) \oplus One[i] \oplus \varphi_{carry}(i)$$

and for every i

$$P'[i] = P[i] \oplus \mathsf{co}_{|P|}Y[i] \oplus \varphi_{carry}(i)$$

□

2.3 Dynamic Reductions

2.3.1 First-Order Reductions

In descriptive complexity, a way of reducing one problem to another problem is the first-order reductions. First-order reductions are used to build reduction that honor dynamic complexity, the bfo-reductions which we will see in the next section. We already know that a first-order query is a first-order definable mapping from structures of one schema to structures of another, so a first-order reduction is simply a first order query that is also a many one reduction. At first, we give an example and then the formal definition.

Example 2.3.1. *The problem Reach*$_d$ *is reducible to Reach*$_u$*. Indeed, given a directed graph, G, let G*$'$ *be the undirected graph that results from G by the following steps:*

(i) *Remove all edges out of t.*

(ii) *Remove all multiple edges leaving any vertex.*

(iii) *make each remaining edge undirected.*

It is very easy to see that there is a deterministic path in G from s to t if and only if there is a path from s to t in G$'$*.*

The first-order reduction is the expression $I_{d-u} = \lambda_{xy}(\varphi_{d-u},s,t)$ *whose meaning is, "Make the new edge relation* $\{(x,y) \mid \varphi_{d-u}\}$*, and map s to s and t to t."*

$$\alpha(x,y) \equiv E(x,y) \wedge x \neq t \wedge (\forall z)(E(x,z) \rightarrow z = y)$$

$$\varphi_{d-u}(x,y) \equiv \alpha(x,y) \vee \alpha(y,x)$$

I_{d-u} leaves the size of the domain unchanged, so we will say that I_{d-u} is a unary first-order reduction. Now we give the definition of the many-one reductions and first-order reductions.

Definition 2.3.2. *Let $\mathcal{C}$ be a complexity class, and let A and B be boolean queries over σ-structures and τ-structures, respectively. Let I be a mapping from σ-structures to τ-structures with the property that for all σ-structures $\mathcal{A}$,*

$$\mathcal{A} \in \mathsf{A} \Leftrightarrow \mathrm{I}(\mathcal{A}) \in \mathsf{B}$$

Then I is C-many-one reduction from A to B. When I is a first order query, it is a first-order reduction.

Example 2.3.3. *In a previous section, we see that the problem Parity and Multiplication are in DynFo. Now we show that there is a first-order reduction from Parity to Multiplication. Let $\tau = \langle A^1,\ B' \rangle$ be the schema of structures that are a pair A, B of boolean strings. Then Mult is query from τ-structures to τ_s-structures, where τ_s is the schema of boolean strings. Since reductions map boolean queries to boolean queries, we actually deal with the boolean version of MULT. MULTb is a boolean query on structures of vocabulary $\tau' = \langle A^1,\ B', c, d \rangle$ that is true iff bit c of the product of A and B is "d".*

The first-order reduction I from τ_s-structures to τ'-structures is given by the the following formulas:

$$\varphi_A(x, y) \equiv y = max \wedge S(x)$$

$$\varphi_B(x, y) \equiv y = max$$

$$I \equiv \lambda_{xy}\langle \mathsf{true}, \varphi_A, \varphi_B, \langle 0, max\rangle, \langle 0, 1\rangle\rangle$$

Observe that the effect of this reduction is to line up all the bits of string $\mathcal{A}$ into column n-1 of the generated product and the end product of those two binary strings give us the number of ones.More specific:

The string A is: 0 ... 0 s_0 0 ... 0 s_1 ... 0 ... 0 s_{n-1} (the length of every "0 ... 0 s_i" is n)

The string B is: 0 ... 0 1 0 ... 0 1 ... 0 ... 0 1 (the length of every "0 ... 0 1" is n)

The product of A and B gives the sum of n rows of the form "0 ... 0 ... s_i 0 ... 0 ... " for every i ∈ [0,n-1], and this sum gives a number P which actually is the number of one's of string $\mathcal{A}$.
It follows that

$$\mathcal{A} \in \mathsf{Parity} \Leftrightarrow \mathrm{I}(\mathcal{A}) \in \mathsf{Mult_b}$$

as desired. Thus, Parity $\leq_{fo}$ Mult$_b$. It follows that if Mult were first-order, then Parity would be as well, which is a contradiction since we already know that Parity is not in Fo.

This I is also a unary first-order reduction, since is leaves the size of the domain unchanged. Now we give the definition of k-ary first-order reductions. these map structures with domain size n to structures with domain size n^k.

Definition 2.3.4. *Let σ and τ be two schemata, with $\tau = \langle R_1^{a_1},\ldots,R_r^{a_r},c_1,\ldots,c_s\rangle$. Let $S \subseteq$ STRUCT[σ], $T \subseteq$ STRUCT[τ] be two problems. Let k be a positive integer. Suppose we are given an r-tuple of formulas $\varphi_i \in \mathcal{L}(\sigma)$, i=1,...,r, where the free variables of φ_i are a subset of $\{x_1,\ldots,x_{k\cdot a_i}\}$. Finally, suppose we are given an s-tuple, $\vec{t} = \{\vec{t_1},\ldots\vec{t_s}\}$, where each $\vec{t_j}$ is a k-tuple of constant symbols from $\mathcal{L}(\sigma)$. Let $I = \lambda_{x_1,\ldots x_d}\langle\varphi_1,\ldots,\varphi_r,\vec{t}\rangle$ be a tuple of there formulas and constants.(Here d = max$_i$(ka$_i$)). Then I induces a mapping also called I from STRUCT[σ] to STRUCT[τ] as follows. Let $\mathcal{A} \in$ STRUCT[σ] and let n be the size of the domain of $\mathcal{A}$.*

$$I(\mathcal{A}) = \langle\{0,\ldots,n^k-1\},R_1,\ldots,R_r,\vec{t}\rangle$$

Here each c_j is given by the corresponding k-tuple of constants. The relation R_i is determined by the formula φ_i, i = 1,...,r.

Suppose that I is a many-one reduction from S to T, i.e, for all $\mathcal{A}$ in STRUCT[σ],

$$\mathcal{A} \in S \Leftrightarrow I(\mathcal{A}) \in T$$

Then we say that I is a k-ary first-order reduction from S to T.

Since the composition of first-order reductions are a first-order reduction, then we have that the relation $\leq_{fo}$ is transitive.

Now, we will give a k-ary first-order reduction [19].

Theorem 2.3.5. *Reach is hard for NL via first-order reductions.*

Proof. Let S $\subseteq$ STRUCT[σ] be a boolean query in NL. Let N be the non-deterministic logspace Turing machine that accepts S. We construct a first-order reduction I : STRUCT[σ] $\to$ STRUCT[τ_g] such that for all $\mathcal{A} \in$ STRUCT[σ],

$$N(bin(\mathcal{A}))\downarrow \quad \Leftrightarrow \quad I(\mathcal{A}) \in Reach \quad (1)$$

Let c be such that N uses at most clogn bits of worktape for inputs bin($\mathcal{A}$), with n = $\|\mathcal{A}\|$. Let $\sigma = \langle R_1^{a_1},\ldots, R_r^{a_r},c_1,\ldots,c_s\rangle$ and let a = max{a_i | 1 $\leq$ i $\leq$

r}. Let k = 1 + a + c. Consider a run of N on input bin($\mathcal{A}$). We code an instantaneous description (ID) of N's computation as a k-tuple of variables:

$$\mathrm{ID} \quad = \quad (p, r_1, ..., r_a, w_1, ..., w_c)$$

The idea is that variables r_1,...., r_a encode where in one of the input relations the read head of N is looking. If for example it is looking at relation R_i, then,

$$\text{N's read head is looking at a " 1 "} \Leftrightarrow \mathcal{A} \models R_i(r_1, ..., r_{a_i})$$

Variables w_1,..., w_c encode the contents of N's work tape. Remember that each variable represents an element of $\mathcal{A}$'s n-element domain, so it corresponds to a logn-bit number. Finally , we need O(loglogn) bits of further information to encode: (1) the state of N, (2) which input relation or constant symbol the read head is currently scanning, and (3) the position of the work head. We assume that n is sufficiently large that all of this information can be encoded into a single variable, p.

Now we start to build the desire k-ary first-order query I and show that it satisfies (1). I will be constructed as follows:

$$\mathrm{I} = \lambda_{\mathrm{ID},\mathrm{ID}'}\langle \mathbf{true}, \varphi_N, \alpha, \omega \rangle$$

where

(i) The domain relation being "**true**" indicates that for any $\mathcal{A} \in$STRUCT[σ], the domain of I($\mathcal{A}$) consists of all k-tuples from the domain of $\mathcal{A}$, $|\mathrm{I}(\mathcal{A})| = |\mathcal{A}|^k$.

(ii) $\mathcal{A} \models \varphi_N(\mathrm{ID},\mathrm{ID}')$ if and only if (ID,ID$'$) is a valid move of N on input bin($\mathcal{A}$),

(iii) $\mathcal{A} \models \alpha(\mathrm{ID}_i)$ if and only if ID_i is the unique initial ID of N , for inputs of size $||\mathcal{A}||$, and,

(iv) $\mathcal{A} \models \omega(\mathrm{ID}_f)$ if and only if ID_f is the unique accept ID of N , for inputs of size $||\mathcal{A}||$.

Formulas α and ω are the following

$$\alpha(x_1, ..., x_k) \quad \equiv \quad x_1 = x_2 = ... = x_k = 0$$

$$\omega(x_1, ..., x_k) \quad \equiv \quad x_1 = x_2 = ... = x_k = max$$

Formula φ_N is not hard, but it is more tedious. It is essentially a disjunction over N's finite transition table.

A typical entry in the transition table $(\langle q, b, w\rangle, \langle q^{'}, i_d, w^{'}, w_d\rangle)$. This says that in state q, looking at bit b with the input head and bit w with the work head, N may go to state $q^{'}$, move its input head one step in direction i_d, write bit $w^{'}$ on its work tape and move its work head one step in direction w_d. The corresponding disjunct in φ_N must decode the old state from variable p and must decode from p which input relation is being read. Say it is R_i. Then the bit b is "1" if and only if $R_i(r_1,...,r_{a_i})$ holds. Similarly, we extract from p the segment j of the work tape that is currently being scanned together with position s on that worktape. Thus, bit w is "1" if and only if $BIT(w_j,s)$ holds.

With these details completed, it now follows that for any $\mathcal{A} \in STRUCT[\sigma]$, $I(\mathcal{A})$ is the computation graph of N on input $bin(\mathcal{A})$. It follows that N accepts $bin(\mathcal{A})$ if and only if there is a path in $I(\mathcal{A})$ from s to t. □

2.3.2 Bfo-Reductions

Now we give reductions that honor dynamic complexity, i.e. reductions for comparing dynamic complexity classes. First-order are too powerful for these classes and that's why we define a restricted version of first-order reductions.

Definition 2.3.6. *Bounded expansion, first-order reductions (bfo) are first-order reductions such that each tuple in a relation and each constant of the input structure affects at most a constant number of tuples and constants in the output structure.. Furthermore, a bfo reduction is required to map the initial structure, $\mathcal{I}$, to a structure with only a bounded number of tuples present. If this condition is relaxed we get bounded expansion, first-order reductions with precomputation (bfo$^+$). If S is reducible to T via bounded expansion, first-order reduction (with precomputation), we write $S \leq_{\mathtt{bfo}} T$ $(S \leq_{\mathtt{bfo}^+} T)$*

Example 2.3.7. *In example 2.2.1, we see that the problem Reach$_d$ is first-order reducible to Reach$_u$. It is easy to see that this first-order reduction is bounded expansion, since each insertion or deletion of an edge (a,b) from the graph G can cause at most two edges to be inserted or deleted in $G^{'} = I_{d-u}(G)$.*

Since the composition of bfo reductions are a bfo reduction, then we have:

Proposition 2.3.8. *The relation $\leq_{\texttt{bfo}}$ is transitive.*

Now we will give the proof that bfo reductions preserve dynamic complexity from [19].

Proposition 2.3.9. *If $T \in$ DynFO and $S \leq_{\texttt{bfo}} T$, then $S \in$ DynFO.*

Proof. We are given a bfo reduction, I, from S to T. If any change to an input, $\mathcal{A}$, for S corresponds to a bounded number of changes to I(A). Given a change operation r to $\mathcal{A}$, we immediately know the number of possibly affected tuples and constants in I(A), let's say k. Since I is first-order, we can existentially quantify all the changed tuple and constants. Then by using the DynFO algorithm for T, we can respond to the at most k changes. Every answer of a query of T will be the correct answer to the given query to, since I is a many-one reduction. □

Example 2.3.10. *In example 2.2.3, we see that the problem Parity is first-order reducible to the problem Multiplication. We can see that this first-order reduction is bounded expansion, since each change of one bit of string $\mathcal{A}$ can cause one change to the string of A in the product $A \times B$. Therefore, Parity is bfo-reducible to Multiplication., and we have already proven that Multiplication in is DynFO, so Parity is in DynFO by Proposition 2.3.9. which is known from Example 2.1.1..*

Example 2.3.11. *Now we will see that the first-order reduction from Theorem 2.3.5. is not bfo reduction. The change of a bit of bin($\mathcal{A}$) from 0 to 1 or from 1 to 0 can affect a non-constant number of edges in I($\mathcal{A}$), since if we change a tuple from a relation then that wil affect the most of the instantaneous descriptions, and that's why I is not a bfo reduction. Also, if I is a bfo reduction and as we will show in Section 4 that Reach $\in$ DynFO, then we have from Proposition 2.3.9. that NL $\subseteq$ DynFO. The conjecture NL $\subseteq$ DynFO has not proven yet, but it is believed that does not hold.*

2.3.3 Logical Truth-Table Reductions

In this section, we define logical truth-table reductions which are a more general notion of reductions between queries than the bfo-reductions from the previous section.

The logical truth-table reductions is what it is the truth table reductions to many-one reductions. The notion of the truth-table reducibility was defined originally by Post [25] and a definition is :

Definition 2.3.12. *There exists a truth-table reduction from problem A to problem B if there is a function which, on input x, precomputes a number of questions about membership in B and describes a Boolean function of the answers to these questions which specifies the membership of x in A [4].*

Here, the idea is to define a reduction from a computation of a query q on a structure $\mathcal{S}$ to the computation of a query q$'$ by

(i) defining in a first-order fashion, from S, a collection of structures of the form $\mathcal{J}(\mathcal{S},\vec{a})$, one for each tuple $\vec{a}$ of some arity m over the domain of $\mathcal{S}$,

(ii) compining all query results $q'(\mathcal{J}(\mathcal{S},\vec{a}))$ into a structure $\mathcal{S}'$, and

(iii) defining $q(\mathcal{S})$ from $\mathcal{S}$ and $\mathcal{S}'$ by a first-order formula.

Here, (i) corresponds to the numbers of questions from Definition 2.3.12, (ii) makes these questions about the membership in B and (iii) corresponds to the Boolean function of the answers to these questions. That's how logical truth-table reductions relate to truth-table reductions.

The structures of the form $\mathcal{J}(\mathcal{S},\vec{a})$ are defined not over the domain of $\mathcal{S}$ but over some Cartesian product over this domain, so we have to deal with two "dimension parameters". One of these two is d which will denote the dimension of the domain of the structures $\mathcal{J}(\mathcal{S},\vec{a})$ and the other is m which will denote the arity of the tuples $\vec{a}$. Now we give the formal definition of first-order truth-table reductions.

Definition 2.3.13. *An interpretation $\mathcal{J}$ of dimension d and arity m from databases with schema σ to databases with schema τ consists of*

- *a σ-formula $\varphi_D(\vec{x},\vec{y})$ and*
- *σ-formula $\varphi_R(\vec{x}_1,\ldots,\vec{x}_{Ar(R)},\vec{y})$, for every $R \in \tau$,*

where $\vec{y} = y_1,\ldots,y_m$, $\vec{x} = x_1,\ldots,x_d$ and, for every j, $\vec{x}_j = x_{j_1},\ldots,x_{j_d}$.

For every σ-structure $\mathcal{S}$ and each m-tuple $\vec{a}$ over the domain S of $\mathcal{S}$, the interpretation $\mathcal{J}$ defines a structure $\mathcal{J}(\mathcal{S},\vec{a})$ with

- *domain $D^{\vec{a}} \overset{\text{def}}{=} \{ \vec{b} \in S^d \mid \mathcal{S} \models \varphi_D(\vec{b},\vec{a}) \}$, and*
- *relations*

$$R^{\vec{a}} \overset{\text{def}}{=} \{(\vec{b}_1,\ldots,\vec{b}_{Ar(R)}) \in (D^{\vec{a}})^{Ar(R)} \mid \mathcal{S} \models \varphi_R(\vec{b}_1,\ldots.,\vec{b}_{Ar(R)},\vec{a})\}$$

, for every R from τ.

A first-order truth-table query-to-query reduction $\mathcal{R} = (\mathcal{J},\varphi)$ from q to q' consists of an interpretation $\mathcal{J}$ and a formula φ with free variables $x_1,\dots,x_k$, where k is the arity of q, which fulfills the following reduction property.

For every structure $\mathcal{S}$, $q(\mathcal{S})$ is the set $\{\vec{t} \mid \mathcal{S}' \models \varphi(\vec{t})\}$, where the structure $\mathcal{S}'$ with domain dom(S) is defined as follows. Let d be the dimension of $\mathcal{J}$, m its arity, l the arity of q', and σ, τ the schemata of $\mathcal{J}$.

- *$\mathcal{S}'$ has all relations from $\mathcal{S}$;*
- *Furthermore, $\mathcal{S}'$ has a relation Q of arity $m + dl$ that contains all tuples of the form $(\vec{a},\vec{s})$, where $\vec{a} \in dom(\mathcal{S})^m$ and $\vec{s} \in q'(\mathcal{J}(\mathcal{S},\vec{a}))$. In $(\vec{a},\vec{s})$, the l-tuple s over universe $dom(\mathcal{S})^d$ is considered a dl-tuple in the obvious way.*

The formula φ is called the wrap-up formula of the reduction.

Now we will give the definition of the bounded expansion of fo-tt reductions from [7] as we did for fo-reductions in the previous section.

Definition 2.3.14. *A fo-tt reduction is a bfo-tt reduction if its interpretation has bounded expansion. The interpretation $\mathcal{J}$ has bounded expansion if there is a constant c such that for all structures $\mathcal{S}_1$, $\mathcal{S}_2$ over the domain D, which differ by exactly one tuple, and for every $\vec{a}$ over D, the databases $\mathcal{J}(\mathcal{S}_1,\vec{a})$ and $\mathcal{J}(\mathcal{S}_2,\vec{a})$ differ by at most c tuples.*

Example 2.3.15. *Now we give a bfo-tt reduction from 2-Sat to Reach. The Boolean query 2-Sat asks whether a given propositional formula in 2-CNF*[5] *has a satisfying assignment. A instance of 2-Sat is represented as a structure The domain of this structure is the set of variables of a formula φ that the structure represents. The clauses of φ are represented by three binary input relations C_{TT}, C_{TF} and C_{FF} such that a tuple $(x,y) \in C_{TT}$ corresponds to a clause $x \vee y$, a tuple $(x,y) \in C_{TF}$ to a clause $x \vee \neg y$, and a tuple $(x,y) \in C_{FF}$ to a clause $\neg x \vee \neg y$. So the change operation of tuples corresponds to a change operation of clauses in a natural way.*

Let ϑ be a 2-CNF formula with set of Variables V and $G = (V \cup \overline{V}, E)$ be a graph where $\overline{V} = \{\neg x \mid x \in V\}$ and E contains the edges $(\neg L,L')$ and $(\neg L',L)$ if $L \vee L'$ is clause in ϑ. So there is reduction that maps ϑ to G and we see that ϑ is satisfiable if and only if there is no variable $x \in V$ such that there exist both a path from $\neg x$ to x and from x to $\neg x$ in G.

[5]A propositional formula is in 2-CNF if it is in conjunctive normal form and each clause contains at most two literals.

Indeed, if ϑ is satisfiable then we assume that there is a variable $x \in V$ such that there exist both a path from $\neg x$ to x and from x to $\neg x$ in G. In the assignment that satisfies ϑ, variable x can be true or false. Let $x \longrightarrow y_1 \longrightarrow \dots \longrightarrow y_k \longrightarrow \neg x$ be the path from x to $\neg x$, then formula ϑ consists of the clauses: $(\neg x \vee y_1)$, $(\neg y_1 \vee y_2)$, $(\neg y_2 \vee y_3)$, $\dots$, $(\neg y_k \vee \neg x)$. If we assign x to true, at least one of these clauses is not satisfied. Indeed, if we assign y_1 to true, then for the rest of the variables we have that y_i=True for $i \in [2,k]$. Thus the clause $(\neg y_k \vee \neg x)$ is false and otherwise, for the rest of the variables we have that y_i=false for $i \in [2,k]$. Thus the clause $(\neg x \vee y_1)$ is false. Similarly, we work if assign x to false, but for the path from $\neg x$ to x. Therefore, we lead to contradiction and so there is no variable $x \in V$ such that there exist both a path from $\neg x$ to x and from x to $\neg x$ in G

Now if there is no variable $x \in V$ such that there exist both a path from $\neg x$ to x and from x to $\neg x$ in G then we assume that ϑ is not satisfiable. For the path from x to $\neq x$, let U be the set of vertices in G reachable from x, Z the set of vertices from which $\neg x$ is reachable and $W := V(G) \setminus (U \cup Z)$. By hypothesis and construction, U, Z, W are pairwise disjoint, and there are no edges from U to $V \cup W$ or from $U \cup W$ to Z . Let me assign the true value to every variable in $U \cup W$ (including x) and the false value to every variable in V (including the literal $\neg x$). It is now simple to check that this formula is satisfied for this truth assignment. Similarly, for path from $\neg x$ to x, we get that ϑ is satisfiable. Therefore, we lead to contradiction and the equivalence holds.

Now about the reduction, the graph G will be encoded over the set of pairs over V. For two variables $u \neq v$ from V, a pair (x,u) will represent x and (x,v) will represent $\overline{x}$. We demand $u \neq v$ because if $u = v$ then these kind of parameters do not contribute to success of the reduction and that's why we ignore 2-CNF formulas with one variable . This can be achieved by a two-dimensional interpretation $\mathcal{J}$ of arity 2. For each pair of variables (u,v), $\mathcal{J}(\vartheta,u,v)$ is the graph defined as above with u and v indicating positive and negative literals, respectively. Thus, the formula $\varphi_D((x_1,x_2), (y_1,y_2))$ could be chosen as $(x_2 = y_1) \vee (x_2 = y_2)$, allowing only pairs in the domain of $\mathcal{J}(\vartheta,u,v)$ whose second entry is one of the parameters given by y_1 and y_2. The formula $\varphi_E((x_{11},x_{12}),(x_{21},x_{22}),(y_1,y_2))$ can be chosen as

$$((C_{TT}(x_{11}, x_{21}) \vee C_{TT}(x_{21}, x_{11})) \wedge (x_{12} = y_2) \wedge (x_{22} = y_1))\vee$$

$$((C_{TT}(x_{11}, x_{21}) \vee C_{TT}(x_{21}, x_{11})) \wedge (x_{12} = y_1) \wedge (x_{22} = y_2)) \vee \cdots$$

where (y_1, y_2) is the parameters for this interpretation and for x_{21} or x_{22} to be equal to y_1 means that their corresponding entry is a positive literal *and it is negative if they are equal to y_2. So this formula says that for the variables x_{11} and x_{21} we have:*

$$((\neg x_{11} \vee x_{21}) \vee (x_{21} \vee \neg x_{11})) \vee ((x_{11} \vee \neg x_{21}) \vee (\neg x_{21} \vee x_{11})) \vee \cdots$$

for every possible match between x_{12}, x_{22} and y_1, y_2.

For the C_{TF} the formula is the following:

$$((C_{TF}(x_{11}, x_{21}) \vee C_{TF}(x_{21}, x_{11})) \wedge (x_{12} = y_2) \wedge (x_{22} = y_1)) \vee$$

$$((C_{TF}(x_{11}, x_{21}) \vee C_{TF}(x_{21}, x_{11})) \wedge (x_{12} = y_1) \wedge (x_{22} = y_2)) \vee \cdots$$

and this formula says that for the variables x_{11} and x_{21} we have:

$$((\neg x_{11} \vee \neg x_{21}) \vee (x_{21} \vee x_{11})) \vee ((x_{11} \vee x_{21}) \vee (\neg x_{21} \vee \neg x_{11})) \vee \cdots$$

for every possible match between x_{12}, x_{22} and y_1, y_2.

Also, for C_{FF} the formula is the following:

$$((C_{FF}(x_{11}, x_{21}) \vee C_{FF}(x_{21}, x_{11})) \wedge (x_{12} = y_2) \wedge (x_{22} = y_1)) \vee$$

$$((C_{FF}(x_{11}, x_{21}) \vee C_{FF}(x_{21}, x_{11})) \wedge (x_{12} = y_1) \wedge (x_{22} = y_2)) \vee \cdots$$

and this formula says that for the variables x_{11} and x_{21} we have:

$$((x_{11} \vee \neg x_{21}) \vee (\neg x_{21} \vee x_{11})) \vee ((\neg x_{11} \vee x_{21}) \vee (x_{21} \vee \neg x_{11})) \vee \cdots$$

for every possible match between x_{12}, x_{22} and y_1, y_2.

Finally, the wrap-up formula φ can be chosen as

$$\exists u, v(u \neq v) \wedge \neg \exists x(Q((x, u), (x, v), (u, v)) \wedge Q((x, v), (x, u), (u, v)))$$

where the relation Q is true if and only if there is path between two vertices and Q has arity 6, since the tuple $\vec{a}$ = (u, v) is of arity 2 and all the 2-tuples s is over the domain dom(ϑ)^2 and thus, the relation Q has arity 2 + 2 × 2 = 6. The formula φ says that there exists a pair of variables (u, v) with u ≠ v,where u and v indicate the positive and negative literals in ϑ, respectively, and that there is no variable x such that there exist both a path from ¬x to x and from x to ¬x in G.

The modification of a single clause in ϑ induces only two first-order definable modifications to the edge set of each corresponding graph, since each clause give two edges to the edge set. Therefore, the reduction is also bounded.

In the previous section, we gave the proof that bfo-reductions preserve the dynamic complexity. We will give the proof of the same about the bfo-tt reductions from [7].

Proposition 2.3.16. *DynFO is closed under bfo-tt reductions, that is, if there is a bfo-tt reduction from a query q to a query q' and $q' \in$ DynFO, then $q \in$ DynFO.*

Proof. Let $(\mathcal{J},\varphi)$ be a bfo-tt reduction from q to q' with expansion bound c. Let σ,τ be the schemata of q and q', respectively, and let d be the dimension and m the arity of $\mathcal{J}$. let $\mathcal{P}'$ be the dynamic program of q'.

The program $\mathcal{P}'$ can be turned into a dynamic program $\mathcal{P}$ for σ-structures that have one auxiliary relation R of arity m + dAr(R'), for every input and auxiliary relation R' of $\mathcal{P}'$. For each m-tuple $\vec{a}$, the $\mathcal{P}$ simulates the behavior of $\mathcal{P}'$ on $\mathcal{J}(\mathcal{S},\vec{a})$, independently. Since the interpretation $\mathcal{J}$ has expansion bound c then a change operation for $\mathcal{S}$ translates into a sequence of at most c change operations for $\mathcal{J}(\mathcal{S},\vec{a})$. So, for evry tuple $\vec{a}$, there is a sequence of at most c change operations of $\mathcal{P}'$, which can be applied successively and in parallel for each $\vec{a}$.

Since the query relation of $\mathcal{P}'$ is one of its auxiliary relations, $\mathcal{P}'$ has, in particular, the relation Q from the reduction property above available, and can therefore compute q($\mathcal{S}$) in a first-order manner. □

For the dynamic complexity class DynFO(+, ×), the reduction that we work with is bfo(+, ×)-tt reduction.

Definition 2.3.17. *A bfo(+, ×)-tt reduction is defined almost the same way as a bfo-tt reduction. The difference is that S has three distinguished relations $\leq$, +, and ×, representing arithmetic on the universe. In such a reduction, the query q must not depend on the choice of $\leq$, +, and ×, but $\mathcal{J}(\mathcal{S},\vec{a})$ of course can.*

By an adaptation of the proof of Proposition 2.2.1., it can be shown that:

Proposition 2.3.18. *If there is a bfo(+, ×)-tt reduction from a query q to a query q', then $q \in$ DynFO(+, ×) if $q' \in$ DynFo.*

2.4 DynFO and Other Complexity Classes

In this thesis, we talk about dynamic complexity and their complexity classes. More precisely, the complexity class that we deal with the most is DynFO. Every time a new class is defined, it is necessary to show where exactly it is compare to other classes. We will show where DynFO stands compare to class like AC^0, P, P-complete.

A way that we have defined DynFO is as the complexity class that consists of all the queries that can be maintained after single tuple insertions and deletions using first-order formulas (or AC^0-circuits). So we have that $AC^0 \subseteq$ DynFO. But, in the previous section, we show that Parity $\in$ DynFo and we are already known that Parity is not in AC^0[7]. Thus, $AC^0 \subset$ DynFO.

It is not difficult to show that DynFO $\subseteq$ P.

Theorem 2.4.1. *DynFO $\subseteq$ P.*

Proof. Let φ be a query over graphs that is maintained in DynFO. Let $\mathcal{I}$ be the structure coding the graph G. We show that the problem whether $\mathcal{I} \models \varphi$ can be solved in polynomial time. Let $\mathcal{A}$ be the auxiliary structure. Let $\mathcal{I}_\emptyset$, $\mathcal{A}_\emptyset$ be the empty input structure and the auxiliary empty structure, respectively, before any insertion or deletion and $\mathcal{I}_i$, $\mathcal{A}_i$ be input structure and the auxiliary structure, respectively, after insertion of i edges. After the first insertion we want to answer the question $\mathcal{I}_1 + \mathcal{A}_1 \models \varphi$, which needs polynomial time to be answered, since only one element has inserted and is easy to check if $\mathcal{I}_1 + \mathcal{A}_1 \models \varphi$ holds. After the second insertion we want to answer the question $\mathcal{I}_2 + \mathcal{A}_2 \models \varphi$, which also needs polynomial time to be answered for the same reason as before. So after the i ($i \leq |E|$, where E is set of edges of G) insertions the question $\mathcal{I}_i + \mathcal{A}_i \models \varphi$ needs polynomial time to be answered. Finally, after the |E| insertion, where $G_{|E|}$=G, the question $\mathcal{I}_{|E|} + \mathcal{A}_{|E|} \models \varphi$ needs polynomial time to be answered. If any deletion happens and from i edges we have i-1, then we already know that the question for i-1 edges can be answered in polynomial time. Therefore, the problem whether $\mathcal{I} \models \varphi$ can be solved in polynomial time, so DynFO $\subseteq$ P. □

There is the conjecture that DynFO is a proper subset of P. Also, the relation between DynFO and P-complete is very interesting. We will give the proof of that DynFO contains P-complete problems from [19] and more specific the padded form of Reach_a. Reach_a is the reachability problem for alternating graphs. Alternating graph is a directed acyclic graph whose vertices are marked "$\vee$" or "$\wedge$". Suppose the a and b are vertices of alternating graph G, and a has edges to the vertices $x_1,\dots,x_n$. We say that b is reachable from a if and only if:

(i) a = b; or

(ii) a is marked "$\wedge$", $n \geq 1$, and b is reachable from all the x_i's; or

(iii) a is marked "$\vee$" and b is reachable from some x_1.

We also note that if all vertices of the graph are marked "$\vee$" then this will be the usual notion of reachability.

As we see at Section 2.3.2, reduction with bounded expansion and precomputation are appropriate reductions for comparing the dynamic complexity of problems. The most natural reductions for P-complete problems are either bounded expansion, or can be modified to be so and the same holds for Reach_a.

Proposition 2.4.2. *Reach_a is P-complete for P via bfo$^+$ reductions.*

Proof. It is shown that Reach_a is complete for ASPACE[logn] via first order reductions[18]. Also it holds that ASPACE[logn] = P [5] [21] and then we have that Reach_a is complete for P via first-order reductions.

Now we want to show that this first-order reduction has bounded expansion. An alternating machine does not look at its input until the last step of its computation, and so each input bit is copied only once and the first-order reductions from [11] has bounded expansion. □

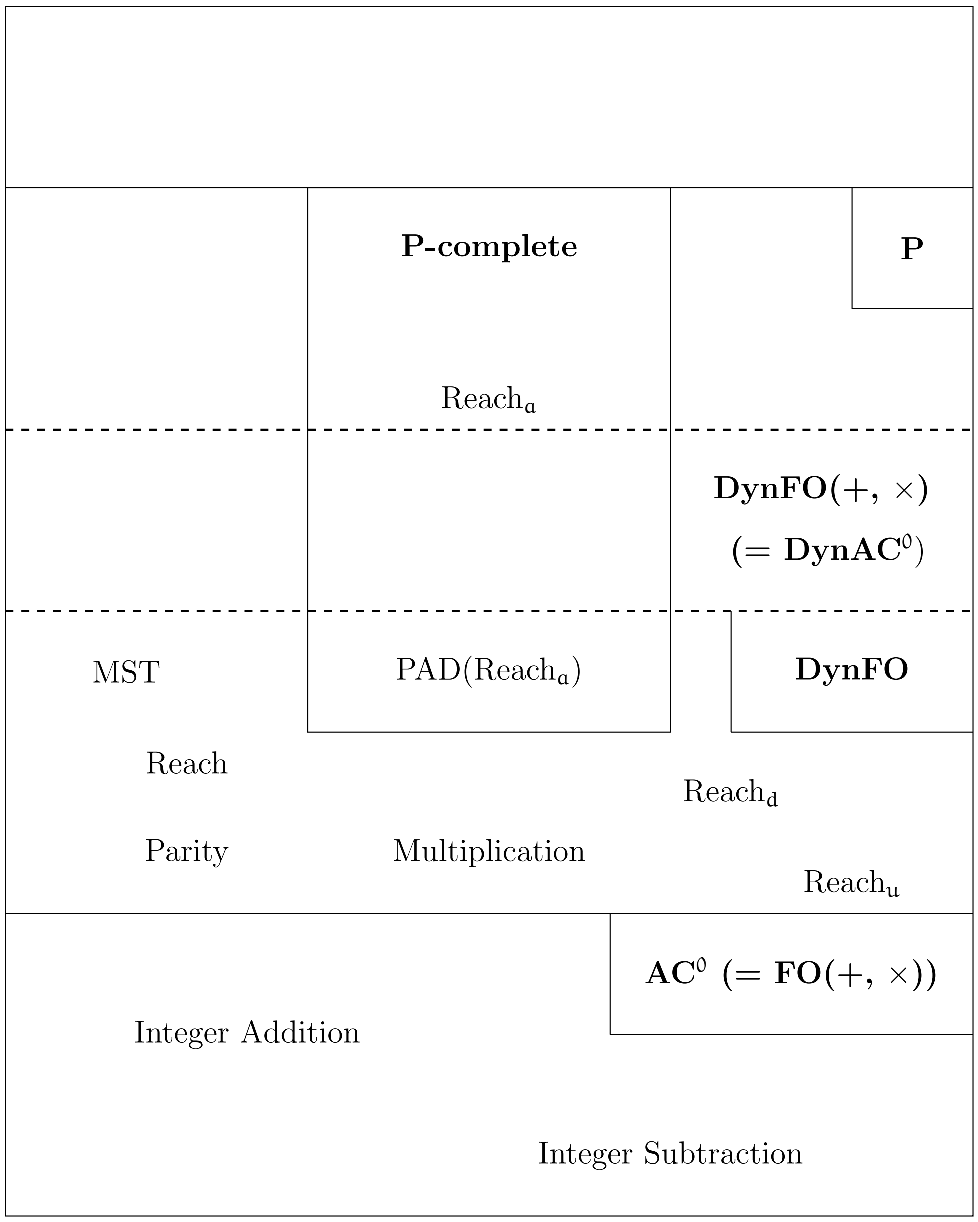
Definition 2.4.3. *For any problem S, define the padded form of S as follows:*

$$\mathsf{PAD}(S) = \{w_1, w_2, ..., w_n \mid n \in \mathbb{N}, w_1 = w_2 = ... = w_n, w_1 \in S\}$$

Theorem 2.4.4. *PAD(Reach_a) is in DynFO.*

For every S, PAD(S) is computationally equivalent to S. From Proposition 2.4.2., we have that Reach_a is P-complete via bounded-expansion first-order reductions with precomputation and so is PAD(Reach_a). Also, for every change operation to an input of S, it required n change operations to the input of PAD(S). Therefore, the DynFO algorithm for PAD(Reach_a) has n first-order steps to respond to any change of the input to S. Reach_a is in FO[n], since FO[$n^{O(1)}$] = P from Theorem 15.5 from [19] and from Proposition 2.4.2. we know that Reach_a is in P. That's why Theorem 2.4.4. holds and we have that a P-complete problem is in DynFO(Figure 2.1).

The complexity class DynFO contains also L-complete and NL-complete problems. The problem Reach_d is a L-complete problem and is in DynFO from Theorem 2.2.4 and Reach is a NL-complete problem that is in DynFO, which we will show it in Section 4.



1)The black line means that one class is proper subset of the above classes.

2)The dashed line means that one class is not for sure a proper subset of the above classes.

Figure 2.1: Hierarchy of Complexity Classes

2.5 DynFO(+, ×) vs. DynFO

In this section, we will give the proof of that if a query has a specific property and belongs to DynFo(+, ×) then also belongs to DynFO from [7]. This property is the weak domain independence.

Definition 2.5.1. *A query q is weakly domain independent, if* $q(\mathcal{D}) \upharpoonright adom(\mathcal{D}) = q(\mathcal{D} \upharpoonright adom(\mathcal{D}))$*, for all databases* $\mathcal{D}$[6].

Example 2.5.2. *The query Reach is a well-known weakly domain independent query. As we have already known, the query representing Reach maps a* τ_g*-structure to a binary relation that contains the transitive closure of the graph. More specifically, the reachability query returns, for a given graph G, all pairs (s,t) of vertices, for which there is a path from s to t. The active domain* $adom(\mathcal{D})$ *is the set of all the vertices in G, that occur in the edge relation. Thus, the query over the database* $\mathcal{D} \upharpoonright adom(\mathcal{D})$ *is equal to the query over the database* $\mathcal{D}$ *that is restricted from* $adom(\mathcal{D})$*, since the elements that occur in the edge relation is the only possible elements for which may exist a path from one to another.*

It is pretty obvious to see that any other variant of Reach, i.e., $Reach_a$, $Reach_u$, *and* $Reach_d$*, are weakly domain independent.*

Example 2.5.3. *Now, we will see a query that is not weakly domain independent. That query is the connectivity query, which returns, for a given graph G, if there is a path for every two vertices of the graph. The active domain* $adom(\mathcal{D})$ *is the set of all the vertices in G, that occur in the edge relation. Thus, the query is not weakly domain independent, since when we restrict* $\mathcal{D}$ *to* $adom(\mathcal{D})$*, we exclude all the isolated vertices and that affects the connectivity of the graph.*

Proposition 2.5.4. *If a query* $q \in$ *DynFO(+, ×) is weakly domain independent, then* $q \in$*DynFO.*

In section 2.2.2., we saw that DynFo(+, ×) = DynAC^0. More precisely, we proved that DynFO$^\$$ = DynAC^0, where DynFO$^\$$ is the same complexity class as DynFO except that we assume the entire domain is activated. Thus, if the domain is activated then we have that a DynFO algorithm has the same expressive power as DynFO(+, ×) and that is the reasoning for the proof of the Proposition 2.5.4.. A way to activate the domain, before any

[6]$\mathcal{D} \upharpoonright$A is the restriction of a database $\mathcal{D}$ over the set A and results from $\mathcal{D}$ by restricting all relations to tuples over A.

change operation starts, is to insert and after delete the edge {u,u} for every element u of the domain. If the domain is activated then the relations $\leq_{\mathtt{ad}}$, $+_{\mathtt{ad}}$ and $\times_{\mathtt{ad}}$ are also in the structure and represent a linear order on the activated elements, and corresponding ternary addition and multiplication relations.

Now we give the proof of the Proposition 2.5.4..

Proof. Let q be a weakly domain independent query and $\mathcal{P}$ a DynFO(+, $\times$) algorithm that maintains q. Here, we assume that q uses only one binary relation E as input relation, the adaptation for arbitrary structures is straightforward. Since $\mathcal{P}$ is a DynFO(+, $\times$) algorithm, it means that any change operation is applied to an initially empty structure and there exist non-empty initial relations that provide a linear order and the corresponding addition and multiplication relations on the domain.

We will construct a DynFO algorithm $\mathcal{P}'$ that simulates $\mathcal{P}$. The main difficulty is that a DynFO algorithm uses initially empty auxiliary relations unlike a DynFO(+,$\times$) algorithm. Therefore, $\mathcal{P}'$ cannot simulate $\mathcal{P}$ right from the beginning of the change sequence, since it does not have $\leq$, + and $\times$ available.

Now we will fully describe algorithm $\mathcal{P}'$.

The update algorithm $\mathcal{P}'$ maintains a linear order $\leq$, an addition relation + and a multiplication relation $\times$ on the set A of the activated elements. This part of the simulation is the same as we have seen in Lemma 2.2.2.. The relation $\leq$ orders the activated elements in the order of activation. For correctness: if an (a,b) is inserted and both a and b were not activated before then a and b become the largest elements of $\leq$ with a $\leq$ b. If only one of a,b were not activated then this elelement becomes the largest of $\leq$. The update formula for determining whether a tuple (x,y) is in the relation $\leq$ after inserting an edge (a,b) into E is the same as the one from the Lemma 2.2.2. . Also that an element x is activated can be expressed by the corresponding formula from Lemma 2.2.2. If a deletion happens, then the update formula for $\leq$ is $\varphi_{\leq}^{\mathrm{dele}}(x,y,a,b) = x \leq y$, since the elements are already activated and so their order is still on. The relation $\leq$ associates A with the set of size $m \overset{\mathrm{def}}{=} |A|$ of the form $[m\text{-}1]_0$[7] with the natural linear order, i.e., the minimum element in $\leq$ is considered as 0, the second as 1, so on and the maximum element in $\leq$ is considered as m-1. We use numbers as constants in formulas. It is very easy and simple to replace there

[7]By [n], we denote the set {1,...,n} and by $[n]_0$, the set {0,1,...,n}.

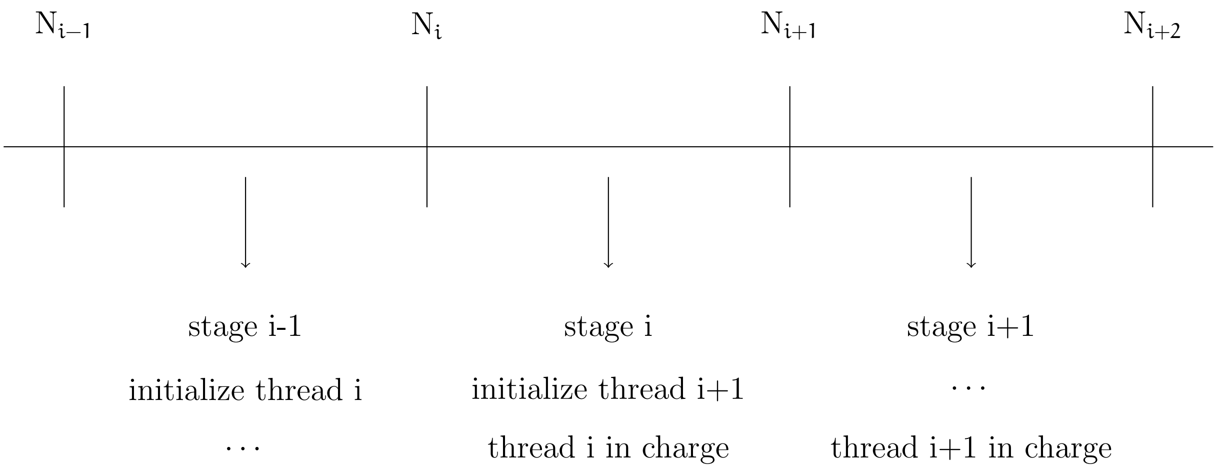


Figure 2.2: Illustration of stages in the proof of Proposition 2.5.3.

numbers by formulas. For example, the subformula $x \geq 1$ can be replaced by $\exists x_1 \exists x_2\ (x_1 \leq x_2) \wedge (x_2 \leq x)$. Furthermore, the addition relation and multiplication relation are maintained as in Lemma 2.2.2..

The main work of this algorithm $\mathcal{P}'$ is to run each computation of $\mathcal{P}$. The way this algorithm is going to do it is by separating each computation in stages, based on the size of A. Let U be the universe of size n, then the i-th stage of the computation of $\mathcal{P}$ starts when more than N_i and at most N_{i+1} elements of U are activated, where $i < \log \log n + 1$ and $N_i = 2^{2^i}$ $(= N_{i-1}^2)$ for every $i \geq 0$. The case, where at most $2 = N_0$ elements are activated, is very easily to dealt with separately as we have seen at Lemma 2.2.2..

Now $\mathcal{P}'$ will simulate the stages of $\mathcal{P}$ by using threads. For each i, thread i is responsible for stage i and begins when the computation of $\mathcal{P}$ is in stage i-1 and ends at the end of stage i of $\mathcal{P}$. During stage i of $\mathcal{P}$, thread i is in charge. The query result for q is always provided by the thread that is in charge(Figure 2.2).

When thread i starts, a linear order, an addition relation, and a multiplication relation over $[N_{i-1}$ - $1]_0$ are available. For these relations, a linear order, an addition relation, and a multiplication relation on 4-tuples over $[N_{i-1}$ - $1]_0$ can be easily defined in first order logic. Thread i considers 4-tuples as 4-digit base-N_{i-1} numbers and thus identifies a 4-tuple (u_1,u_2,u_3,u_4) over $[N_{i-1}$ - $1]_0$ with the number $u_1\times N_{i-1}^3 + u_2\times N_{i-1}^2 + u_3\times N_{i-1} + u_4$. As we have said before, the relations $<$, $+$, $\times$ can be lifted to relations over 4-tuples in first-order logic as we have seen in Theorem 2.1.1. and, thus, we do not have to maintain them as separate auxiliary relations.

Thread i uses the set of 4-tuples over $[N_{i-1}$ - $1]_0$ as domain of size N_{i+1} ($(2^{2^{i-1}})^4 = N_{i-1}^4 = N_i^2 = N_{i+1}$), since every structure has to have in its domain all elements that can be used in a relation. Thus, the domain $([N_{i-1}$ - $1]_0)^4$ of size N_{i+1} is large enough to represent each new element that is activated by some tuple over $[N_{i-1}$ - $1]_0$. For that reason, during stage i-1 and i, thread i maintains a bijection g_i between the activated elements and 4-tuples over $[N_{i-1}$ - $1]_0$. At the start of thread i, $g_i(k) = (0,0,0,k)$, for every $k \in [N_{i-1}$ - $1]_0$, and g_i is extended in a straightforward fashion. For example, if the last activated domain corresponds to the number $N_{i-1} + 3$ then we have that $g_i(N_{i-1} + 3) = (0,0,1,3)$. Also, it uses one 4k-ary auxiliary relation $R^{'}$ for every auxiliary or input k-ary relation R of $\mathcal{P}$. It starts on the structure over $([N_{i-1}]_0)^4$ with empty relation E, with the linear order and the corresponding addition and multiplication relations over $([N_{i-1}$ - $1]_0)^4$, but otherwise empty auxiliary relations. So now $\mathcal{P}^{'}$ can simulate $\mathcal{P}$ on an initially empty structure.

Let $E^{'}$ be the auxiliary 8-ary relation corresponding to the binary relation E of $\mathcal{P}$. At the beginning of stage i-1 at least N_{i-1} elements are activated and so the relation E contains almost N_{i-1}^2 edges and the corresponding relation $E^{'}$ is empty, since thread i has not started yet. Therefore, thread i cannot simulate $\mathcal{P}$ until it catches up with $\mathcal{P}$. So, thread i needs to make sure that at the end of stage i-1, all tuples in E have corresponding tuples in $E^{'}$.

As we have said, thread i has to catch up with $\mathcal{P}$ and for that it needs to add more than one edge per step to $E^{'}$ and more specific at most four edges per step. The reason will be given below. Thus, thread i has to catch up until the end of the stage i-1, so at the beginning of stage i, its relation $E^{'}$ and relation E are isomorphic to each other under g_i. That's why we get the symmetric difference of E and $g_i^{-1}(E^{'})$, $\Delta \overset{\mathrm{def}}{=} E \triangle g_i^{-1}(E^{'})$, where $g_i^{-1}(E^{'})$ is a subset of the 2-tuples of the activated elements that give through g_i the corresponding elements of $E^{'}$, and the goal is to decrease the symmetric difference to zero and then thread i would have caught up with $\mathcal{P}$.

If a change δ occurs in $\mathcal{P}$, then it is applied to E without triggering the associated update operations. Afterwards, thread i identifies the lexicographically smallest four pairs e_1, e_2, e_3, e_4 over $[N_i]_0$ in Δ and then for each one applies the appropriate update operations as follows:

(i) If $e_j \in E \setminus g_i^{-1}(E')$, thread i simulates the operation of $\mathcal{P}$ for an insertion of e_j and thus algorithm $\mathcal{P}'$ inserts the element $g_i(e_j)$ in E'.

(ii) If $e_j \in g_i^{-1}(E') \setminus E$, thread i simulates the operation of $\mathcal{P}$ for a deletion of e_j and thus algorithm $\mathcal{P}'$ deletes the element $g_i(e_j)$ from E'.

The change δ can be an insertion or a deletion. If it is an insertion, after the work of thread i $|\Delta|$ decreases by three tuples and if it is a deletion, $|\Delta|$ decreases by five tuples. Therefore, we can say as a conclusion that $|\Delta|$ decreases by at least three tuples, unless $|\Delta| < 4$ already. At the beginning of the stage i-1, relation E might already contain up to N_{i-1}^2 edges, whereas E' is empty and so Δ may contain at most N_{i-1}^2, i.e., $|\Delta| \leq N_{i-1}^2$. We conclude that after each step $|\Delta|$ decreases by at least three tuples and thus, $\frac{1}{3}N_{i-1}^2$ change steps suffice for thread i to catch up. Stage i-1 has at least $\frac{1}{2}(N_i - N_{i-1})$ change steps, since the minimum amount of change steps corresponds to each change step to be an insertion and exactly 2 elements to be activated with each insertion. Also, we have that

$$N_i - N_{i-1} = N_{i-1}^2 - N_{i-1} \geq \frac{3}{4}N_{i-1}^2 \text{ for } i \geq 2,$$

since

$$i \geq 2 \;\Leftrightarrow\; i-1 \geq 1 \;\Leftrightarrow\; N_{i-1} \geq N_1 \;\Leftrightarrow\; N_{i-1} \geq 4 \;\Leftrightarrow\; N_{i-1}^2 \geq 4N_{i-1} \;\Leftrightarrow$$
$$\frac{1}{4}N_{i-1}^2 \geq N_{i-1} \;\Leftrightarrow\; N_{i-1}^2 - \frac{3}{4}N_{i-1}^2 \geq N_{i-1} \;\Leftrightarrow\; N_{i-1}^2 - N_{i-1} \geq \frac{3}{4}N_{i-1}^2,$$

then stage i-1 has at least $\frac{1}{2} \times \frac{3}{4}N_{i-1}^2 = \frac{3}{8}N_{i-1}^2$, which is larger than $\frac{1}{3}N_{i-1}^2$ and thus, it is possible for thread i to catch up. When $i = 1$ and $|\Delta| \leq 3$, we work the same as before.

If thread i instead of identifying the four smallest pairs of edges, identify one, two or three, then the change steps that will suffice for thread i to catch up, it would not be enough as we saw above.

During stage i, thread i can simulate $\mathcal{P}$ in a lockstep fashion, mimicking every step. More precisely, when a change δ modifies a tuple e, thread i of algorithm $\mathcal{P}'$ applies δ to $g_i(e)$, and performs the necessary updates to the relations of thread i. If Q' is the auxiliary relation in $\mathcal{P}'$ that corresponds to query relation Q of $\mathcal{P}$, then $g_i^{-1}(Q')$ is the query result during stage i.

After stage i, thread i is abandoned and thread i+1 is in charge. Algorithm $\mathcal{P}'$ will maintain a counter for each thread and when the counter of thread i reaches the value $(N_{i-1})^4$ ($= N_{i+1}$) then the thread i+1 has to take over in the next step.

This completes the description of how the thread i works, for each i.

In the previous paragraphs, we saw how the thread i works and an important question is whether each thread will use its own auxiliary relations. The answer is no and the way we will handle this is instead of using different relations for each thread, we can increase the arity of each relation by one and the extra component will be used to identify the thread, for which it is used. For example, all the 8-ary relation E' are encoded into one 9-ary relation $\overline{E}$ and the relation E' of thread is the set of tuples

$$\{\overline{t} \ \mid (i, \overline{t}) \in \ \overline{E}\}.$$

Another example is the bijection g_i that can be encoded into one 6-ary relation that contains the tuple (i, k,$\overline{t}$) if and only if $g_i(k) = \overline{t}$.

Now we will show the correctness of algorithm $\mathcal{P}'$. Let α denote some change sequence in $\mathcal{P}$ and α_i the prefix of α until the end of stage i. Let γ_i denote the change sequence of $\mathcal{P}$ that arises from the process that reduces the size of Δ for thread i during stage i-1 as we have explained above. We argued above that $\alpha_{i-1}(\mathcal{I}_\emptyset) = \gamma_{i-1}(\mathcal{I}_\emptyset)$, where $\mathcal{I}_\emptyset$ is the empty input structure of algorithm $\mathcal{P}$. The program $\mathcal{P}'$ applies $g_i(\gamma_{i-1})$ in stage i-1, and therefore $g_i(\mathcal{P}_{\gamma_{i-1}}(\mathcal{I}_\emptyset, \mathcal{A}_\emptyset)) = \mathcal{P}'_{g_i(\gamma_i)}(g_i(\mathcal{I}_\emptyset, \mathcal{A}_\emptyset))$ is easy to be shown by an induction on the length of change sequences. Thus, the input and auxiliary structures of thread i at the end of stage i-1 is the isomorphic image of the input and auxiliary structure that $\mathcal{P}$ can reach for the input database $\alpha_i(\mathcal{I}_\emptyset)$.

Finally, we can see that during stage i, $\mathcal{P}'$ can keep track of the changes and updates. As we saw before, thread i uses a domain of size N_{i+1}, where $N_{i+1} < n$, and not of size n. This does not affect the output of $\mathcal{P}'$, since the query q is weakly domain independent and so the activated part of the domain during stage i is always of size at most N_{i-1}. Thus, $\mathcal{P}'$ has a correct output, at anytime. □

Chapter 3

Linear Algebra and Matrices

In the next section, we will give the proof of that the reachability problem on directed graphs is in DynFO from [7]. A very important contribution to this proof is a linear algebra problem SVRank. As far as we have known, there are not computational linear algebra problems that have been studied in dynamic complexity, except from the Boolean Matrix Multiplication [16]. Thus, the main problem for linear algebra problems in dynamic complexity is how to represent the problem. If for example we have the problem matrix rank, then the key question is how to represent the numbers that appear in the matrix. The important thing is to represent all the elements of a matrix, and for that we use a representation that does not allow matrices with large numbers but suffices for applications in which matrix entries are not larger than the number of rows in the matrix.

At first, we will give the definition of some notation about the matrices from [7] that will be very useful later. Let A be a matrix, the entry in the i-th row and j-th column of A will be denoted as A[i,j] and the i-th entry of vector x as x[i]. By $e_i^{(n)}$, we denote the n-dimensional unit(column) vector e with e[i]=1 and e[j]=0 for $j \neq i$. The rank and the determinant of a matrix A are denoted by rank(A) and det(A), respectively. For a prime number p, we denote by $\text{rank}_p(A)$, the rank of A as a matrix over $\mathbb{Z}_p$. A $(m \times m)$ matrix A over $\mathbb{Z}$ has small values, if for each i, j $\in \{1,...,m\}$, $|A[i,j]| \leq m$.

The query SVRank returns the rank of the matrix A, where A is a $(m \times m)$-matrix with small values.The query representing SVRank maps τ-structures to structures that consist of a unary relation Q that is supposed to contain a unique element r, the rank of A. Let τ-structure, where τ is arbitrarily, be the structures that represent $(m \times m)$ matrices A. The domain of those structures contain m+1 elements and there is a linear order $<$ that give us a total ordering of D and so D is equivalent to the set $[m]_0$. There are compatible + and × relations as well. Furthermore, those structures consist of two ternary relations A_+, A_- to represent the entries of A. That A[i,j] = a, for a ∈ {1,...,m} is represented by a triple (i,j,a) in A_+. Similarly, if A[i,j] = a, for a ∈ {-m,...,-1}, there is a triple (i,j,a) in A_-. For each i,j, there is at most one triple (i,j,a) in $A_+ \cup A_-$. If, for some i,j, there is no such triple (i,j,a),then A[i,j] = 0.

Change operations might insert a triple (i,j,a) to A_+ or A_- (in case no (i,j,b) is there) or delete a triple. That is, basically, single-matrix entries can be set to 0 or from 0 to some other value. However, the relations <, + and × cannot be changed.

Chapter 4

Reachability Is in DynFO

In this chapter, we will show that Reachability is in DynFO and the proof of this theorem has four steps. The first step involves the problem SVRank that we defined in chapter 3 and a bfo(+, ×)-tt reduction from Reach to SVRank. The second step is a bfo-tt reduction from SVRank to RankModP, which is a new problem that we will define later. The third step is to show that RankModP is in DynFO. From the first three steps we have that Reach is in DynFO(+, ×) and from Example 2.5.2 we know that Reach is a weakly domain independent query, then we have that Reach is DynFO from Proposition 2.5.4..

In the next three sections, we will give the proof of the three steps separately.

4.1 From Reachability to Matrix Rank

Now we will give the reduction between Reach and SVRank.

Theorem 4.1.1. *There is a bfo(+, ×)-tt reduction from Reach to SVRank.*

Proof. Let G be graph with n vertices and A_G its adjacency matrix, and let s, t be vertices of G. For the matrix I - $\frac{1}{n}A_G$ [17], we know that is is invertible and its inverse is

$$\left(I - \frac{1}{n}A_G\right)^{-1} = I + \sum_{i=1}^{\infty}\left(\frac{1}{n}A_G\right)^{i}. \quad (1)$$

Instead of dealing with the matrix I - $\frac{1}{n}A_G$, we will work with the integer matrix B $\overset{\text{def}}{=}$ nI - A_G, which is also invertible.

If the vertex t is reachable from the vertex s in G and the path from s to t is of length k, where k is arbitrary, then the matrix $(A_G)^k$ has a non-zero entry at the position (s,t). From the equation (1), we will have that the inverse of B has a non-zero entry at position (s,t). Thus, we have that if t is reachable from s then the inverse of B has a non-zero entry at position (s,t).

If the inverse of B has a non-zero entry at position (s,t), this means that $\sum_{i=1}^{\infty}(\frac{1}{n}A_G)^i$ has at least a non-zero entry at position (s,t) and let k be the number for which the sum get the non-zero entry. Thus, we have that the matrix $(A_G)^k$ has a non-zero entry at position (s,t) and so, there is a path of length k from to s to t or t is reachable from s.

Therefore, we have the following:

$$\boxed{\text{t is reachable from s} \Leftrightarrow (B^{-1})[s,t] \neq 0 \qquad (1)}$$

The equivalence (1) is proven above.

$$\boxed{(B^{-1})[s,t] \neq 0 \Leftrightarrow (B^{-1}e_t^{(n)})[s] \neq 0 \qquad (2)}$$

The equivalence (2) is pretty obvious.

$$\boxed{\begin{array}{c} (B^{-1}e_t^{(n)})[s] \neq 0 \\ \Leftrightarrow \qquad (3) \\ \text{Let x be the vector } B^{-1}e_t^{(n)} \text{ and we know that x[s]} \neq 0 \end{array}}$$

The equivalence (3) is pretty obvious.

$$\boxed{\begin{array}{c} \text{Let x be the vector } B^{-1}e_t^{(n)} \text{ and we know that x[s]} \neq 0 \\ \Leftrightarrow \qquad (4) \\ \text{The equation } Bx = e_t^{(n)} \text{ has no solution vector x with x[s]} = 0 \end{array}}$$

The equivalence (4) is pretty obvious.

The equation $Bx = e_t^{(n)}$ has no solution vector x with x[s] = 0

$$\Leftrightarrow \qquad (5)$$

The system $\begin{cases} Bx = e_t^{(n)} \\ (e_s^{(n)})^T x = 0 \end{cases}$ has no solution vector x at all

The proof of the equivalence (5) is the following:

If the equation $Bx = e_t^{(n)}$ has no solution vector x with x[s] = 0 and we can see that for the second equation of the system every solution has x[s] = 0, then the system has no solution vector x.

If the system has no solution vector x, then $Bx = e_t^{(n)}$ has no solution vector with x[s] = 0 because every solution of $(e_s^{(n)})^T x = 0$ has x[s] = 0.

The system $\begin{cases} Bx = e_t^{(n)} \\ (e_s^{(n)})^T x = 0 \end{cases}$ has no solution vector x at all

$$\Leftrightarrow \qquad (6)$$

$e_t^{(n+1)}$ is not in the column space of B^{+s}

B^{+s} denotes the ((n+1)×n)- matrix that is obtained from B by adding as additional row $(e_s^{(n)})^T$. The proof of the equivalence (6) is the following:

From linear algebra, we already know that the multiplication of B^{+s} with $x = [x_1 \ldots x_n]^T$ is equal to a linear combination of the columns of B^{+s} with coefficients the elements of x, i.e.,

$$(x_1 \times \text{the first column of } B^{+s}) + \ldots. + (x_n \times \text{the } n\text{–th column of } B^{+s}).$$

If the system has a solution, then $e_t^{(n+1)}$ is equal to the multiplication of B^{+s} with x and thus, will be in the column space of B^{+s}. Therefore, if the system has no solution then $e_t^{(n+1)}$ is not in the column space of B^{+s}.

If $e_t^{(n+1)}$ is not in the column space of B^{+s}, then the system has no a solution, because otherwise $e_t^{(n+1)}$ will be in the column space of B^{+s}, as we have seen above.

$$\boxed{\begin{array}{c} e_t^{(n+1)} \text{ is not in the column space of } B^{+s} \\ \\ \Leftrightarrow \qquad\qquad (7) \\ \\ B^{+st} \text{ has rank n+1} \end{array}}$$

B^{+st} denotes the extension of B^{+s} by the additional column vector $e_t^{(n+1)}$. The proof of the equivalence (7) is the following:

Since B is invertible, then B and B^{+s} have rank n and since $e_t^{(n+1)}$ is not in the column space of B^{+s}, then B^{+st} has rank n+1.

If B^{+st} has rank n+1 and we know that the matrix B^{+s} has rank n, then this means that $e_t^{(n+1)}$ is not in the column space of B^{+s} because otherwise B^{+st} would have rank n.

From the equivalences (1) - (7), we have that

t is reachable from s $\Leftrightarrow$ B^{+st} has rank n+1

Now we will show how the above equivalence give us a bfo(+, $\times$)-tt reduction from Reach to SVRank.

At the end of the equivalence, we get the ((n+1)$\times$(n+1))-matrix B^{+st} and every value of this matrix is at most n and thus, B^{+st} has small values as we wanted to be from chapter 3.

In the presence of arithmetic, B^{+st} can be obtained from G by a two-dimensional and binary bfo(+, $\times$)-tt reduction ($\mathcal{J}$,φ). For each database $\mathcal{D}$, which represents a graph G, and each pair (s,t) over the domain U of $\mathcal{D}$, $\mathcal{J}(\mathcal{D}$,(s,t)) is a database that encodes B^{+st}. The interpretation $\mathcal{J}$ uses two dimensions because the domain representing B^{+st} is of size n + 2 for graphs with n vertices. A pair (u_1,u_2) from the domain of $\mathcal{J}$ will represent the number $u_1\times$ n + u_2. For example, the number k < n will be represented by (0,k) and the number n + 1 will be represented by (1,1). For each pair (s,t) of vertices, $\mathcal{J}(\mathcal{D}$,(s,t)) is the matrix defined as above with s and t indicating the column vectors that will be added to B so that we get B^{+st}.

For every pair (s,t), the result relation SVRank ($\mathcal{J}(\mathcal{D}$, (s,t))) is the set {(r)}, where r is the rank of B^{+st} and is represented by 2-tuple of the form (u_1,u_2) as above. The relation Q has arity 4, since the tuple $\vec{a}$ = (s,t) is a 2-tuple and all the 1-tuples s are over the domain dom($\mathcal{D}$)^2 and, thus, the arity of the relation Q is 2 + 2 $\times$ 1 = 4. Therefore, the relation Q of arity

4 consists of all 3-tuples (s,t,r), for which r is the rank of B^{+st}. Thus, for each (s,t), the wrap-up formula φ only needs to check whether r = n + 1, where n is the number of vertices of G:

$$\exists s, t Q(s, t, (1, 1)).$$

Finally, each change in G results in only one change in B^{+st}, for every (s,t), since if we have an insertion or a deletion of an edge in G, then we change one entry of the adjacency matrix and thus, one change to B and B^{+st}. Therefore, the reduction actually has expansion bound 1. □

4.2 From Rank to Rank Mod p

As we have already seen, all input matrices for the SVRank query have small entries, but the maintenance algorithm on with the DynFO algorithm will be based needs to deal with large entries. For that reason, we need to find a way to avoid this complication, and that's why we introduce another problem, such that any dynamic algorithm that maintains it, will not have to deal with large numbers. This problem is the following:

RankModp
Input: (m × m)-matrix A with values from {0,1,...,p-1}, prime
$p \leq m^2$
Output: Rank of A over $\mathbb{Z}_p$

The bound m^2 for prime p over RankModp might appear a bit arbitrary, but we will see that it just suffices. We show next that, to maintain the rank of a matrix A with small values, it suffices to maintain its rank over the field $\mathbb{Z}_p$, for sufficiently many primes p. We denote this rank by $\text{rank}_p(A)$. Now we will give a bfo-tt reduction by SVRank to RankModp.

Theorem 4.2.1. *There is a bfo-tt reduction from SVRank to RankModp.*

Proof. At first, we show that rank(A) ≥ k if and only if $\text{rank}_p(A) \geq k$ for some prime $p \leq m^2$. The reduction from SVRank to RankModp is actually pretty simple.

From linear algebra, we already know that rank(A) ≥ k if and only if there is some k × k submatrix A' of A with $\det(A') \neq 0$. The $\det(A')$ is bounded by $m! \cdot m^m$. Indeed, from Hadamard's inequality we have that

$$\det(A') \leq \prod_{i=1}^{m} \|x_i\|,$$

where $x_i = [x_{i_1}\ x_{i_2}\ \cdots\ x_{i_m}]^T$ is the i-th column of A'. Since A is a matrix with small values, i.e., $x_{i_k} \leq m$ for every $k \in [1,m]$, we have that

$$||x_i|| = \sqrt{x_{i_1}^2 + x_{i_2}^2 + \cdots + x_{i_m}^2} \leq \sqrt{m^2 + m^2 + \cdots + m^2} =$$

$$= \sqrt{m^3} = m^{\frac{3}{2}}, \text{ for every } i \in [1,m].$$

Thus,

$$\det(A') \leq \prod_{i=1}^{m} ||x_i|| \leq \prod_{i=1}^{m} m^{\frac{3}{2}} \leq m^{\frac{3}{2} \cdot m} = m^m \cdot m^{\frac{m}{2}}.$$

Furthermore, we have that

$$(m!)^2 = (1 \cdot 2 \cdots m) \cdot (1 \cdot 2 \cdot \cdots \cdot m) =$$

$$(1 \cdot 2 \cdots m-1) \cdot (m \cdot 1) \cdot (2 \cdot \cdots \cdot m) =$$

$$(1 \cdot 2 \cdots m-2) \cdot (m \cdot 1) \cdot (2 \cdot m-1) \cdot (3 \cdot \cdots \cdot m) =$$

$$= \ldots =$$

$$(1 \cdot m) \cdot (2 \cdot m{-}1) \cdot (3 \cdot (m{-}2)) \cdot \cdots \cdot (m{-}2 \cdot 3) \cdot (m{-}1 \cdot 2) \cdot (m \cdot 1).$$

For everyone parenthesis from above holds that is clearly larger than m and the amount of these parentheses are m, so we have that

$$(m!)^2 \geq m^m \Leftrightarrow m! \geq m^{\frac{m}{2}}.$$

Therefore,

$$\det(A') \leq m^m \cdot m^{\frac{m}{2}} \leq m! \cdot m^m.$$

Now, we know that for large enough n, there are more than $\frac{n}{\log n}$ prime numbers between 1 and n from (3.5) of Colorrary 1 from [28]. Thus, for m^2, there are more than $\frac{m^2}{2\log m}$ prime numbers below m^2 and so their product is larger than $m!m^m$, since from [28] and inequality 3.16 we know that

$$\vartheta(x) > x(1 - \frac{1}{\log x}),$$

where $\vartheta(x)$ is the logarithm of the product of all primes $\leq$ x. Thus, for m^2 we have that

$$\vartheta(m^2) > m^2(1 - \frac{1}{\log m^2}) > m \log m$$

and we can see that $\log(m! \cdot m^m) = O(m \log m)$, so the product of all prime numbers below m^2 is clearly larger than $m! \cdot m^m$. Therefore,

$$\det(A') \leq m! \cdot m^m < \prod_{p < m^2 + 1,\ p \text{ is prime}} p.$$

Let $\det(A') \neq 0$. If $\det(A')$ is equal to 0 modulo every prime p below m^2, then we have that $\det(A')$ is equal to the product of all these primes. This is a contradiction because of all that we proved above. Therefore, $\det(A') \neq 0$ if and only if there exists a prime $p \leq m^2$ such that $\det(A') \neq 0 \pmod{p}$. The other direction of the equivalence is pretty obvious. Thus, for large enough m, $\text{rank}(A) \geq k$ if and only if there exists a prime $p \leq m^2$ such that $\text{rank}_p(A) \geq k$.

Then, we give the bfo-tt reduction between SVRank and RankModp. The bfo-tt reduction from SVRank to RankModp consists of the interpretation $\mathcal{J}(\mathcal{D}, \vec{i})$ of dimension 1 and arity 2 and the wrap-up formula φ. The database $\mathcal{D}$ represents an input matrix A for SVRank and the pair $\vec{i} = (i_1, i_2)$ over m is interpreted as the number $n(\vec{i}) = (i_1 - 1)m + (i_2 - 1)$. If $n(\vec{i})$ is not a prime number, then the matrix $B^{\vec{i}} = \mathcal{J}(\mathcal{D},\vec{i})$ is the all-zero matrix and if $n(\vec{i})$ is a prime number, then the matrix $B^{\vec{i}} = \mathcal{J}(\mathcal{D},\vec{i})$ represents the matrix A over $\mathbb{Z}_p$. Thus, the formula $\varphi_D(x, (i_1,i_2))$ could be

$$\exists n([n = (i_1 - 1) \times m + (i_2 - 1)] \wedge prime(n)),$$

where the formula prime(n) is

$$1 < n \wedge \forall u, v(n = u \times v \rightarrow u = 1 \vee v = 1).$$

For every pair $\vec{i}$, the result relation $\text{RankModp}(\mathcal{J}(\mathcal{D},\vec{i}))$ is the set $\{(k)\}$, where k is rank of a matrix over the field $\mathbb{Z}_p$, for $p = n(\vec{i})$. Therefore, the relation Q has arity 3 and consists of all 3-tuples $(\vec{i},k)$, for which k is the rank of the matrix over $\mathbb{Z}_p$. The wrap-up formula φ simply computes the maximum k, such that for some prime $p = n(\vec{i})$, the result relation $\text{RankModp}(\mathcal{J}(\mathcal{D},\vec{i}))$ contains k and the formula is

$$\exists i_1, i_2 Q(i_1, i_2, k).$$

□

4.3 Maintaining Rank Mod p in DynFO

As we have said at the beginning of this section, our goal is to prove that Reach is in DynFO. That's why we have showed the two reductions from sections 4.1 and 4.2, i.e., Reach is bfo(+, ×)-tt reducible to SVrank and SVrank is bfo-tt reducible to RankModp. Now we will show that RankModp is in DynFO and because of Proposition 2.3.16 , then SVRank will be in DynFO and because of Proposition 2.5.4 and that Reach is a weakly domain independence query, then Reach will be in DynFO.

So we want to show that the rank of a matrix modulo a prime can be maintained in DynFO.

Theorem 4.3.1. *RankModp is in DynFO.*

Proof. The algorithm is an adaptation of a dynamic algorithm that has been stated in [13].

Let A be a matrix over $\mathbb{Z}_p$, where $p < m^2$, with small values and we want to maintain the rank of this matrix after a change of A[i, j], for any i, j $\leq$ m. As we have seen in Chapter 3, change operations for a matrix might insert a triple (i,j,a) to A_+ or A_- (in case no (i,j,b) is there) or delete a triple. That is, basically, single-matrix entries can be set to 0 or from 0 to some other value. Instead of maintaing the rank of A, we will maintain an invertible matrix B and a matrix E which is in reduced row-echelon form such that BA = E, i.e. we find a matrix B for which holds that if we multiply it to A from the left, we get a matrix, E, in reduced row-echelon form and we want to maintain these two matrices.

A matrix E is in reduced row-echelon form means that

- the leftmost non-zero entry (the leading entry) in every row is 1,
- the column of such a leading entry only contains zero entries otherwise, and
- rows are sorted according to the position of the leading entry if this is 1, i.e., if row i has as a leading entry, 1, in position j then every other row $i' < i$ must have as a leading entry, 1, in some position $j' < j$. In particular, all rows consisting of only 0's are at the bottom of the matrix.

The rank of E is equal to the rank of A, since B is invertible and we know that if B is invertible then rank(BA) = rank(A) [11]. The rank of A equals the number of non-zero rows of E thanks to rank(E) = rank(A) and the structure of E. Indeed, since E is in reduced row-echelon form then none of its rows can be a linear combination of the others and if the rank of A is lesser than m then the only way the rank of E can be lesser than m is to have rows that consist of only 0's. So if we want to maintain the rank of A, we have to find the number of rows of that consist of only 0's. Thus, maintaining the matrices B and E suffices to maintain the rank of A.

Now, we will describe how to maintain these matrices after a change A[i, j], for any $i, j < m$. We note now that every computation of matrix entries is modulo p. Let A' denote the new matrix after the change of A[i, j] and we will explain how the new matrices B' and A' can be obtained such that $B'A' = E'$.

After a change of A[i, j], the product BA' differs from BA at most in column j, which is pretty obvious from the definition of multiplication of matrices. Thus, to get the desired matrix E' in reduced echelon form, we can proceed as follows:

(i) If column j has more than one leading entry of BA'

- let some entry with a maximum number of successive zeros in its row (right after column j) be the new leading entry
- set this leading entry to 1, and set all other entries of column j to zero , by appropriate row operations.

(ii) If a former leading entry of row k is lost in column j by the change operation in A or by step (i),

- set its new leading entry to 1, i.e., the next non-zero entry in row k and some column $\ell > j$, and set all others entries of column ℓ to 0, by the appropriate row operations.

(iii) If needed, move the at most two rows, for which the position of the leading has changed compared with E to their correct row positions.

As we have said , we want to maintain matrices B and E, so that we get the matrices B' and E' such that $B'A' = E'$ and E' to be in reduced row-echelon form. Since the above description gives E', then we applied the same row operations to B, and this ensures us that $B'A' = E'$. By applying all these row operation to B corresponds to multyplying a suitable elementary matrix from the left to B, so B remains invertible as we have seen in [22]. An illustration of the modifications necessary for one change in matrix A for p = 5 is the following:

At first, for a given A, we find B such that BA = E and E is in reduced row-echelon form. So we have that

$$\begin{array}{ccccc} \mathsf{B} & & \mathsf{A} & & \mathsf{E}\ (\mathrm{mod}\ p) \\ \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 3 & 0 \\ 3 & 0 & 0 & 0 & 1 \end{bmatrix} & \times & \begin{bmatrix} 4 & 0 & 3 & 0 & 0 \\ 0 & 2 & 4 & 0 & 0 \\ 4 & 0 & 3 & 1 & 0 \\ 0 & 2 & 4 & 0 & 2 \\ 3 & 0 & 1 & 0 & 0 \end{bmatrix} & = & \begin{bmatrix} 1 & 0 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Then we change the entry of A, A[1,2], from 0 to 1 and we have that

$$\begin{array}{ccccc} \mathsf{B} & & \mathsf{A}' & & \mathsf{B}\ \times \mathsf{A}'\ (\mathrm{mod}\ p) \\ \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 3 & 0 \\ 3 & 0 & 0 & 0 & 1 \end{bmatrix} & \times & \begin{bmatrix} 4 & [1] & 3 & 0 & 0 \\ 0 & 2 & 4 & 0 & 0 \\ 4 & 0 & 3 & 1 & 0 \\ 0 & 2 & 4 & 0 & 2 \\ 3 & 0 & 1 & 0 & 0 \end{bmatrix} & = & \begin{bmatrix} 1 & 4 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 4 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 3 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Then, we apply the step (i) to B $\times$ A$'$ and B and we have that

$$\begin{array}{ccccc} \mathsf{B\ after\ Step\ (i)} & & \mathsf{A}' & & \mathsf{B}\ \times \mathsf{A}'\ \mathsf{after\ Step\ (i)} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 2 \\ 4 & 3 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 2 & 0 & 3 & 0 \\ 1 & 0 & 0 & 0 & 2 \end{bmatrix} & \times & \begin{bmatrix} 4 & 1 & 3 & 0 & 0 \\ 0 & 2 & 4 & 0 & 0 \\ 4 & 0 & 3 & 1 & 0 \\ 0 & 2 & 4 & 0 & 2 \\ 3 & 0 & 1 & 0 & 0 \end{bmatrix} & = & \begin{bmatrix} 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Then, we apply the step (ii) to B after Step (i) and B $\times$ A$'$ after Step (i) and we get

$$\underset{\text{B after Step (ii)}}{\begin{bmatrix}1&2&0&0&4\\2&4&0&0&4\\0&0&1&0&2\\0&2&0&3&0\\1&0&0&0&2\end{bmatrix}} \times \underset{A'}{\begin{bmatrix}4&1&3&0&0\\0&2&4&0&0\\4&0&3&1&0\\0&2&4&0&2\\3&0&1&0&0\end{bmatrix}} = \underset{\text{B} \times A' \text{ after Step (ii)}}{\begin{bmatrix}1&0&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\\0&1&0&0&0\end{bmatrix}}$$

Finally, we apply Step (iii) to B after Step (ii) and B $\times$ A$'$ after Step (iii)

$$\underset{\text{B after Step (iii)}}{\begin{bmatrix}1&2&0&0&4\\1&0&0&0&2\\2&4&0&0&4\\0&0&1&0&2\\0&2&0&3&0\end{bmatrix}} \times \underset{A'}{\begin{bmatrix}4&1&3&0&0\\0&2&4&0&0\\4&0&3&1&0\\0&2&4&0&2\\3&0&1&0&0\end{bmatrix}} = \underset{\text{B} \times A' \text{ after Step (iii)}}{\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{bmatrix}}$$

Therefore, the matrix B $\times$ A$'$ after Step (iii) is in reduced row-echelon form and so B$'$ = B after Step (iii) and E$'$ = B $\times$ A$'$ after Step (iii).

Now we will give the dynamic algorithm $\mathcal{P}$ that will put RankModp in DynFO:

At first, there is available the logical representation of the matrix A with the relations $\leq$, $+$ and $\times$. Also, $\mathcal{P}$ has auxiliary relations that encode the matrices B and E. After a change of A[i, j], the steps (i)-(iii), described above, can be translated into maintaining rules in first-order logic. so that $\mathcal{P}$ maintains the auxiliary relation that encodes B and E and this will give us matrices B$'$ and E$'$ such that B$'$A$'$ = E$'$. Each of the steps (i)-(iii) can be performed in constant parallel time. This is the way we maintain the rank of A in first-order logic.

$\square$

Chapter 5

More Problems in DynFO

In the previous chapter we presented the proof of Reach is in DynFO. Now using this result and the techniques that developed for proving it, we will give the proof several other problems being in DynFO. At first, from Example 2.3.15 we have that 2-Sat is bfo-tt reducible to Reach and from Proposition 2.3.16, we have that 2-Sat is in DynFO, since Reach is in DynFO.

Corollary 5.0.1. *2-Sat is in DynFO.*

Now are going to study the dynamic complexity of matching in graphs. In this chapter, we will give the proof of PerfectMatching and Maxmatching are in non-uniform DynFO from [7].

Definition 5.0.2. *Non-uniform DynFO is the same dynamic complexity class as DynFO, except that the auxiliary relations may be initialized arbitrary. A query is in non-uniform DynFO, if there is a dynamic algorithm such that, for every finite domain, the auxiliary relations can be chosen in some way such that the algorithm correctly maintains the query.*

MaxMatching
Input: An undirected graph G
Output: The size k of a maximum matching of G

PerfectMatching
Input: An undirected graph G
Output: Does G have a perfect matching?

It has already been proven that MaxMatching and PerfectMatching are non-uniform DynTC^0 [6], i.e, a non-uniform dynamic algorithm with TC^0-updates and is still open whether these two problems are in DynFO.

The idea of [7] is to prove a correspondence between the size of a maximum matching and the rank of a matrix. More precisely, the matrix that the author of [7] used is the Tutte matrix and from an undirected graph we get its Tutte matrix (or T_G as we will denote it, where G is the undirected graph) as follows:

$$t_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in E \text{ and } i < j \\ -x_{ij} & \text{if } (i,j) \in E \text{ and } i > j \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

where the x_{ij} are indeterminates. At first, we will show how to go from the maximum matching to the rank of the Tutte matrix. The next theorem is from [26].

Theorem 5.0.3. *Let G be a graph with a maximum matching of size m. Then rakn(T_G) = 2m.*

Proof. We prove the theorem in two parts:

(i) rank(T_G) $\geq$ 2m. Choose any matching of size m, and let $U \subseteq V$ be the set of vertices matched by it. Let G' be the subgraph of G induced on U and so G has a perfect matching of size m. Since $T_{G_{UU}}$ is the Tutte matrix of G', then $\det(T_{G_{UU}}) \neq 0$[1]. Since $|U| = 2m$, rank(T_G) $\geq$ 2m.

(ii) rank(T_G) $\leq$ 2m. Suppose rank(T_G) = k. Let $T_{G_{ij}}$ be an invertible matrix submatrix of T_G, with $|\ i\ | = |\ j\ | = k$. We want to show that G has a matching of size at least $\frac{k}{2}$. If i = j, then we are done since the restriction of G to the vertices of i must have a perfect matching by Tutte's Theorem [26][2].

Since $\det(T_{G_{ij}}) \neq 0$ and $|\ i\ | = |\ j\ | = k$, by Frobenius's theorem [26], we have that $\det(T_{G_{ii}}) \neq 0$. Since i = i, then as above G has a matching of size $\frac{k}{2}$. Thus, if T_G has rank greater than 2m, then G has matching greater than m which is a contradiction. □

[1]Let A be a n × n matrix, and i and j be subsets of {1,2,...,n}. Then, A_{ij} will denote the submatrix of A obtained by choosing the rows of A corresponding to indices in i and columns corresponding to indices in j.

[2](Tutte's Theorem)Let G be a graph, and let T_G be its Tutte matrix, then $|A| \neq 0$ $\Leftrightarrow$ there exist a perfect matching in G.

Now, we would like to use the rank maintenance algorithm from Section 4.3, but the problem is that the entries of this matrix are indeterminates and by applying that algorithm we would get polynomials with exponential terms. We can solve this problem, by replacing the indeterminates of T_G with positive integer values. For a graph G, let w be a function that assigns a positive integer weight to every edge (i,j) and let $B_{G,w}$ be the integer matrix obtained from T_G by replacing x_{ij} with $2^{w(i,j)}$.

Now, we will see another theorem that will give us a way from maximum matching to the rank of $B_{G,w}$.

Theorem 5.0.4. *If G is graph with a maximum matching of size m and w is a weight assignment for the edges of G, then rank($B_{G,w}$) $\leq$ 2m. Furthermore, if G has a maximum matching with unique minimal weight with respect to w, then rank($B_{G,w}$) = 2m.*

The proof of this theorem uses another theorem which is:

Theorem 5.0.5. *([23], Lemma 2). Let G be a graph with a perfect matching and w a weight assignment such that G has a unique perfect matching with minimal weight with respect to w. Then det($B_{G,w}$) $\neq$ 0.*

Proof. (Theorem 5.0.4.) We know that the rank of a matrix is equal to the size of the largest submatrix with non-zero determinant. Thus, rank($B_{G,w}$) $\leq$ rank(T_G) and therefore, rank($B_{G,w}$) $\leq$ 2m by Theorem 5.0.3..

Now we want to show that if G has a maximum matching of unique minimal weight with respect to w then rank($B_{G,w}$) $\geq$ 2m. We will adapt the proof of Theorem 5.0.3. Let U be the set of vertices contained in the unique maximum matching of G with minimal weight, and G' the subgraph of G induced by U. Observe that G' has a unique minimal weight perfect matching with respect to w. Restricting $B_{G,w}$ to rows and columns labeled by elements from U yields the matrix $B_{G',w'}$, where w' is the weight function restricted to edges from G'. However, det($B_{G',w'}$) $\neq$ 0 by Theorem 5.0.5. and, therefore, rank($B_{G,w}$) $\geq$ 2m since the matching is of size m and so the matrix $B_{G',w'}$ is (2m $\times$ 2m)-matrix. □

Now, our goal is to obtain weighting functions w_1,....,w_{n^2} with weights in [4n], such that for every graph G over [n] there is an $i \in [n^2]$ such that G has a maximum matching with unique minimal weight with respect to w_i. To accomplish this we will follow the technique of [27] and at first, we need the Isolation Lemma as stated at Lemma 11.5 in [20].

Lemma 5.0.6. *(Isolation Lemma) Let m, $M \in \mathbb{N}$ and let $\mathcal{F} \subseteq 2^{[m]}$ be a non-empty set of subsets of [m]. If a weight function $w \in [M]^{[m]}$ is uniformly chosen at random, then with probability at least 1 - $\frac{m}{M}$, the minimum weight subset in $\mathcal{F}$ is unique; where the weight of a subset $F \in \mathcal{F}$ is $\sum_{i \in F} w(i)$.*

We will give another version of the Isolation Lemma, the Non-Uniform Isolation Lemma as stated in [27].

Lemma 5.0.7. *(Non- Uniform Isolation Lemma) Let $m \in \mathbb{N}$ and let $\mathcal{F}_1, \ldots, \mathcal{F}_{2^m} \subseteq 2^{[m]}$ be an enumeration of non-empty sets of subsets of [m]. There are weight functions $w_1, \ldots, w_m$ from $[4m]^{[m]}$ such that for any $i \in [2^m]$ with $\mathcal{F}_i \neq \emptyset$, there exists a $j \in [m]$ such that the minimum weight subset of $\mathcal{F}_i$ with respect to w_j is unique.*

Proof. We call a collection $u_1, \ldots, u_m$ of weight functions bad for some $\mathcal{F}_i$ if no $F \in \mathcal{F}_i$ is a minimum weight subset with respect to any u_j. For each $\mathcal{F}_\rangle \neq \emptyset$, the probability of a randomly chosen weight sequence $U \overset{\text{def}}{=} u_1, \ldots, u_m$ to be bad is at most $(\frac{1}{4})^m$, since if M = 4m from Lemma 5.0.6. we have that the minimum weight subset in $\mathcal{F}_i$ is unique with probability 1 - $\frac{m}{4m}$ and so for each $\mathcal{F}_i$, $1 \leq i \leq m$, the opposite happens with probability $(1 - (1 - \frac{1}{4}))^m = (\frac{1}{4})^m$. Thus, the probability that such a U is bad for some $\mathcal{F}_i$ is at most $2^m \times (\frac{1}{4})^m = (\frac{1}{2})^m < 1$, since each $\mathcal{F}_i$ is a subset of $2^{[m]}$. Hence there exists a sequence U which is good for all $\mathcal{F}_i$. □

We get the following corollary:

Corollary 5.0.8. *Let $G_1, \ldots, G_{2^{n^2}}$[3] be some enumerations of the graphs on [n] and let $\mathcal{F}_1, \ldots, \mathcal{F}_{2^{n^2}}$ be their respective sets of perfect matchings. There is a sequence $w_1, \ldots, w_{n^2}$ of weight assignments assigning a value from $[4n^2]$ to the edges $[n^2]$ such that for every graph G over [n] there is some $i \in [n^2]$ such that if G has a perfect matching then it also has a perfect matching with unique minimal weight with respect to w_i.*

After all this we saw, we conclude that for maintaining the size of maximum matchings of a graph G over [n], we will maintain the rank of matrix B_{G,w_i} for all $i \in [n^2]$. The rank of T_G is the maximum rank among those ranks thanks to Theorem 5.0.4..

Theorem 5.0.9. *PerfectMatching and MaxMatching are in non-uniform DynFO.*

[3]The number of graphs is 2^{n^2}, since G is over [n] and so the number of edges is at most n^2.

Proof. We know that a perfect matching is a special case of a maximum matching and we will only show that MaxMatching is in non-uniform DynFO.

The non-uniform bfo-tt reduction that we will use is the reduction from MaxMatching to RankModp and since RankModp is in DynFO and from an adaptation of Proposition 2.3.18. we have that MaxMatching is non-uniform DynFO.

The procedure is to advice a dynamic algorithm with weighting functions $w_1,...,w_{n^2}$ that assign weights such that for all graphs with n vertices there is a maximum matching with unique minimal weight as we have seen at Corollary 5.0.8. Now every graph with n vertices has a maximum matching with unique minimal weight with respect to w_i and so that will give us the rank of B_{G,w_i} for some w_i by Theorem 5.0.4. Thus, Theorem 5.0.4. provides us with the equivalence between the maximum matching of a graph G and the rank of corresponding matrix B_{G,w_i} fro some w_i.The advise is given to the dynamic algorithm via the initialisation of the auxiliary relations. The algorithm then maintains the ranks for the matrices B_{G,w_i} and outputs the maximal such rank.

Recall that the weight functions assign values of up to $4n^2$ and that, therefore, the determinant of each B_{G,w_i} can be size up to $n! \cdot (2^{4n^2})^n$. Indeed, we have a $(n \times n)$-matrix with values from $\{0,1,...,2^{4n^2}-1\}$ and if we follow the procedure for the determinant from Theorem 4.2.1., then we will get that upper bound for the determinant. Then, it holds that $n! \cdot (2^{4n^2})^n \leq 2^{5n^3}$, and thus it is sufficient to maintain the rank of those matrices modulo up to $5n^3$ many primes, which are contained in the first n^4 numbers for large enough n by the prime number theorem. Indeed, from the prime number theorem we know that the number of primes up to n^4 is equal to $\frac{n^4}{\log n^4} = \frac{n^4}{4\log n}$ and for large enough n holds that $\frac{n^4}{4\log n} > 5n^3$.

The dynamic program computes, for each of the weighting functions w_i and each prime $p \leq n^4$, the rank of B_{G,w_i} modulo p as it did in Section 4.3.

The non-uniform reduction $(\mathcal{J},\varphi)$ from MaxMatching to RankModp is two-dimensional and 6-ary. Two dimensions are used to encode graphs as matrices as described in Chapter 3. For parameter $(p_1,...,p_6)$, the interpretation $\mathcal{J}$ maps a given graph to an instance of RankModp that asks for the rank of B_{G,w_i} mod p where w_i is encoded by the first two parameters and p is encoded by the remaining four parameter, since the amount of weighting function is n^2 and a prime can be up to $5n^3$. For example, for the rank of $B_{G,w_{n+4}}$ mod $4n^3 + k$, where $4n^3 + k$ is a prime, then the parameter for the interpretation $\mathcal{J}$ will be ((1,4),(4,0,0,k)). Another example is the rank of $B_{G,w_{n^2}}$ mod $5n^2 + kn$, where $5n^2 + kn$ is a prime, then the parameter for the interpretation $\mathcal{J}$ will be ((n,0),(0,5,k,0)). For converting edges to entries of B_{G,w_i} mod p, the reduction uses non-uniform relations since every weighting function has different size.

For every pair $(u_2,u_1,p_1,p_2,p_3,p_4)$, where (u_2,u_1) corresponds to the index of the weighting function and (p_1,p_2,p_3,p_4) corresponds to the prime p, the result relation RankModp($\mathcal{J}(\mathcal{D},(u_2,u_1,p_1,p_2,p_3,p_4))$) is the set {(k)},where k is the rank of $B_{G,w_{u_2\times n+u_1}}$. Therefore, the relation Q consists of all 7-tuples $(u_2,u_1,p_1,p_2,p_3,p_4,k)$, for which k is the rank of $B_{G,w_{u_2\times n+u_1}}$. Thus, for each $(u_2,u_1,p_1,p_2,p_3,p_4)$, the wrap-up formula φ determines the highest rank of all these instances. $\square$

Chapter 6

Conclusion and Further Work

In this thesis, we presented and explained with many examples the notion of dynamic complexity. We worked mostly with the dynamic complexity class DynFO and the related reductions. We presented the proof that the reachability query can be maintained in DynFO. The proof uses computational linear algebra problems such as the rank of the matrix and the rank of matrix over $\mathbb{Z}_p$. As an immediate consequence of the result for reachability, 2-satisfiability can also be maintained in DynFO. By combining the linear algebraic part of the proof that reachability is in DynFO with the Isolation Lemma(Lemma 5.0.6), we presented the proof how the size of a maximum matching can be maintained in DynFO with non-uniform initialisation.

In this thesis, we only studied the modification of one tuple for a problem. In the paper with title **"Reachability and Distances under Multiple Changes"** [10], it is showed that Reachability and Distances problems can be maintained in DynFO(+, ×) under changes affecting $O(\frac{\log n}{\log \log n})$ vertices, for graphs with n vertices. At first, the result that Reachability is in DnyFO has been advanced into a very powerful tool: for showing that a query can be maintained in DynFO, it essentially suffices to show that it can be maintained for logn many change steps after initializing the auxiliary data by an AC^1 precomputation [9], where n is the size of the database's (active) domain n. This tool has been successfully applied to show that all queries expressible in monadic second order logic can be maintained in DynFO on structures of bounded treewidth.

Those new techniques motivate a new attack on a larger number of changes. But updating a query after a number of changes that replaces the whole database by a new database is essentially equivalent to the static evaluation problem with built-in relations: the stored auxiliary data has to be helpful for every possible new database, and therefore plays the role of built-in relations. Thus changes should be restricted in some way. The one approach come to mind is to only allow changes of restricted size and this approach gave the next theorem [10].

Theorem 6.0.1. *Reachability can be maintained in DynFO(+, ×) under changes that affect* $O(\frac{\log n}{\log\log n})$ *verticess of a graph, where* n *is the number of vertices of the graph.*

The distance query was shown to be in DynFO+Maj (or DynTC^0) by Hesse [15], where the class DynFO+Maj allows to specify updates with first-order formulas that may include majority quantifiers (equivalently, updates can be specified by uniform TC^0 computations). Then, Hesse's result is generalized to changes of size polylogarithmic in the size of the domain.

Theorem 6.0.2. *[10] Reachability and Distance can be maintained in DynFO+Maj(+, ×) under changes that affect* $O(\log^c n)$ *vertices of a graph, where* $c \in \mathbb{N}$ *is fixed and* n *is the number of vertices of the graph.*

A question for further research in dynamic complexity is whether Distances are in DynFO. The above approach sheds some light on this question. It can be adapted so as to maintain information within DynFO(+, ×) from which shortest distances can be extracted in FO+Maj(+, ×).

Theorem 6.0.3. *[10] Distances can be defined by a FO+Maj(+, ×) query from auxiliary relations that can be maintained in DynFO(+, ×) under changes that affect* $O(\frac{\log n}{\log\log n})$ *vertices.*

In the paper with title **"Dynamic complexity of Reachability: How many changes can we handle?"** [8], the authors extended these results by showing that, for changes of polylogarithmic size, first-order update formulas suffice for maintaining (1) undirected reachability, and (2) directed reachability under insertions. As we have seen above, a subset of the authors showed that the reachability problems can be maintained in DynFO+Maj(+, ×) under changes that affect $O(\log^c n)$ vertices of a graph (Theorem 6.0.2). In this paper the authors make progress on handling changes of polylogarithmic size in DynFO by attacking the challenge from two directions. First, they establish two restrictions for which reachability can be maintained under these changes.

Theorem 6.0.4. *Reachability can be maintained in DynFO(+, ×) under*

- *insertions of polylogarithmically many edges; and*

- *insertions and deletions of polylogarithmically many edges if the graph remains undirected.*

A question for further research is whether reachability for classes of directed graphs can be maintained in DynFO$(+, \times)$ under insertions and deletions of polylogarithmic size. Candidate classes are graphs with bounded treewidth, and directed acyclic graphs.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[2] Miklós Ajtai. Σ_1^1 -formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48, 1983.

[3] David A Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

[4] Samuel R Buss and Louise Hay. On truth-table reducibility to sat. *Information and Computation*, 91(1):86–102, 1991.

[5] Ashok K Chandra and Larry J Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 98–108. IEEE, 1976.

[6] Samir Datta, William Hesse, and Raghav Kulkarni. Dynamic complexity of directed reachability and other problems. In *International Colloquium on Automata, Languages, and Programming*, pages 356–367. Springer, 2014.

[7] Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in dynfo. *Journal of the ACM (JACM)*, 65(5):1–24, 2018.

[8] Samir Datta, Pankaj Kumar, Anish Mukherjee, Anuj Tawari, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of reachability: How many changes can we handle? *arXiv preprint arXiv:2004.12739*, 2020.

[9] Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *arXiv preprint arXiv:1704.07998*, 2017.

[10] Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. *arXiv preprint arXiv:1804.08555*, 2018.

[11] Harry Dym. *Linear algebra in action*, volume 78. American Mathematical Soc., 2013.

[12] Kousha Etessami. Dynamic tree isomorphism via first-order updates to a relational database. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 235–243, 1998.

[13] Gudmund Skovbjerg Frandsen and Peter Frands Frandsen. Dynamic matrix rank. *Theoretical computer science*, 410(41):4085–4093, 2009.

[14] Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, 1984.

[15] William Hesse. The dynamic complexity of transitive closure is in dyntc0. *Theoretical Computer Science*, 296(3):473–485, 2003.

[16] William Hesse and Neil Immerman. Complete problems for dynamic complexity classes. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 313–322. IEEE, 2002.

[17] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[18] Neil Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384–406, 1981.

[19] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012.

[20] Stasys Jukna. *Extremal combinatorics: with applications in computer science*. Springer Science & Business Media, 2011.

[21] Dexter Kozen. On parallelism in turing machines. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 89–97. IEEE, 1976.

[22] Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 71. Siam, 2000.

[23] Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354, 1987.

[24] Sushant Patnaik and Neil Immerman. Dyn-fo: A parallel, dynamic complexity class. *Journal of Computer and System Sciences*, 55(2):199–209, 1997.

[25] Emil L Post. Recursively enumerable sets of positive integers and their decision problems. *bulletin of the American Mathematical Society*, 50(5):284–316, 1944.

[26] Michael O Rabin and Vijay V Vazirani. Maximum matchings in general graphs through randomization. *Journal of algorithms*, 10(4):557–567, 1989.

[27] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000.

[28] J Barkley Rosser, Lowell Schoenfeld, et al. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6(1):64–94, 1962.

[29] Nicole Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Transactions on Computational Logic (TOCL)*, 6(3):634–671, 2005.