Complete Problems for Dynamic Complexity Classes

William Hesse*
Computer Science Dept.
UMass, Amherst, USA
www.cs.umass.edu/∼whesse

Neil Immerman*
Computer Science Dept.
UMass, Amherst, USA
www.cs.umass.edu/∼immerman

Abstract

We present the first complete problems for dynamic complexity classes including the classes Dyn-FO *and* Dyn-ThC^0*, the dynamic classes corresponding to relational calculus and (polynomially bounded) SQL, respectively. The first problem we show complete for* Dyn-FO *is a single-step version of the circuit value problem (SSCV).*

Of independent interest, our construction also produces a first-order formula, ζ, that is in a sense universal for all first-order formulas. Since first-order formulas are stratified by quantifier depth, the first-order formula ζ emulates formulas of greater depth by iterated application. As a corollary we obtain a fixed quantifier block, QBC, that is complete for all first-order quantifier blocks.

1 Introduction

Traditionally complexity theory has focused on static problems. All standard complexity classes are defined in terms of the complexity of checking – upon presentation of an entire input – whether the input satisfies a certain property.

For many applications of computers (including databases, text editors, program development) it is more appropriate to model the process as a dynamic one. There is a large set of data that is repeatedly modified by users over a period of time. A dynamic algorithm is a method of storing and modifying data in a data structure, so that each change and query may be performed very efficiently.

There is an extensive literature in dynamic algorithms and amortized analysis; but with the exception of [MSV94], a theory of dynamic complexity had been lacking. In [PI97], Patnaik and Immerman began the development of a dynamic complexity theory from the descriptive point of view.

*Research supported by NSF grant CCR-9877078.

In this paper we consider a simpler and more general approach to dynamic complexity than in [PI97]. Rather than consider dynamic versions of static problems, we consider the general problem of processing a sequence of operations. Previous work on Dyn-FO considered only static functions of an input that was subject to simple updates; we consider any dynamic, continuing computation in our model. The complete problems we have found are a result of expanding the model of dynamic complexity in this way.

We find that it is natural to think of a dynamic problem as a sequence of regular languages, $A_1 \subseteq \Sigma_1^\star, A_2 \subseteq \Sigma_2^\star, A_3 \subseteq \Sigma_3^\star \ldots$, where each Σ_n is a polynomial-size set of operations.

This results in a definition of Dyn-$\mathcal{C}$ for any (static) complexity class $\mathcal{C}$. These definitions are consistent with, but slightly extend the corresponding definitions in [PI97]. It is also natural from this point of view to define a dynamic reduction as a uniform sequence of bounded homomorphisms.

In [PI97] the complexity class, Dynamic First-Order Logic (Dyn-FO) was studied. This is the class of requests that admit algorithms such that each request can be performed as a first-order definable operation on the current data structure. It was shown in [PI97] that many interesting properties are in Dyn-FO including all regular languages, majority, multiplication, undirected graph reachability, graph reachability for acyclic graphs, transitive reduction for acyclic graphs, minimum spanning tree, bipartiteness, k-edge connectivity, etc.

More recently, Hesse showed that the dynamic version of the general graph reachability problem (DynREACH) is in Dyn-ThC^0 [Hes01]. It thus also follows that many linear algebra problems including matrix inversion and determinant are in Dyn-ThC^0. This is particularly interesting as Dyn-ThC^0 is the class of dynamic problems computable in (polynomially bounded[1]) SQL [Imm99, Th 14.9].

[1]We say "polynomially bounded" to disallow the creation of new domain elements. Otherwise, an SQL algorithm might create exponentially many new elements, letting it decide problems that it couldn't when the database size remains polynomially bounded, cf. [LW99].

Using our new formulation of dynamic computation, we present the first complete problems for dynamic complexity classes including Dyn-FO and Dyn-ThC^0. Such problems had long been sought, cf. [PI97].

The first problem that we show complete for Dyn-FO is a single step circuit value problem (SSCV). SSCV is complete in a natural way for Dyn-FO because any fixed first-order formula can be simulated by a uniform sequence of AC^0 circuits [Imm99, Th. 5.22].

However, SSCV, is not complete via the weakest version of reductions that we describe, because these weak reductions do not have the power to produce such AC^0 circuits before the computation begins.

To overcome this problem we construct a first-order formula ζ which is in a sense complete for all first-order formulas. Of course, Sipser proved that there is a strict alternation hierarchy for first-order formulas [Sip83]. Thus there can be no single first-order formula that is truly complete for all first-order formulas. What we mean is that every first-order formula, φ, can be simulated by the formula ζ iterated $d+1$ times where d is the depth of φ, evaluated on a structure that is a polynomial padding of the original structure, i.e.,

$$\mathcal{A} \models \varphi \qquad \Leftrightarrow \qquad \text{PAD}(\varphi, \mathcal{A}) \models \zeta^{(d+1)} \ .$$

An equivalent way of stating this is that we have constructed a complete first-order quantifier block (QBC).

Using ζ, we define the problem CSSCV which is the same as SSCV except that it is initialized with the first-order definable AC^0 circuits corresponding to ζ. We show that CSSCV is complete for Dyn-FO via our weakest reductions: bounded first-order homomorphisms.

This paper is organized as follows: In §2 we provide the necessary background, in §3 we introduce our new formulation of the class of dynamic problems, in §4 we explain our model of dynamic computation, in §5 we define the resulting dynamic complexity classes, in §6 we explain our model of dynamic reductions, in §7 we define the SSCV problem and prove that it is complete for Dyn-FO, in §8 we construct a universal first-order formula, ζ, and prove as a corollary that there exists a complete first-order quantifier block, we then use ζ to construct the problem CSSCV which is Dyn-FO-complete via our weakest reductions, in §9 we use our main result to construct other Dyn-FO-complete problems including a monotone version of CSSCV, and an iterated Boolean matrix multiplication problem. We also include some complete problems for other classes including Dyn-ThC^0. Finally, in §10 we describe directions for further research.

2 Background on Descriptive Complexity

We briefly describe the needed background in descriptive complexity [Imm99]. A vocabulary, $\tau = \langle R_1^{a_1}, \ldots, R_t^{a_t}, c_1, \ldots, c_s \rangle$, is a tuple of relation symbols and constant symbols. The superscripts of the relation symbols denote their arities. A *structure* with vocabulary τ is a tuple, $\mathcal{A} = \langle |\mathcal{A}|, R_1^{\mathcal{A}} ... R_t^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}} \rangle$ whose universe is the nonempty set $|\mathcal{A}|$. We will assume throughout that $|\mathcal{A}| = \{0, 1, \ldots, n-1\}$ and reserve $n = \|\mathcal{A}\|$ for the cardinality of the universe of a given structure $\mathcal{A}$. We let STRUC$[\tau]$ denote the set of finite structures of vocabulary τ. Let,

$$\text{STRUC}_n[\tau] \qquad = \qquad \left\{ \mathcal{A} \in \text{STRUC}[\tau] \;\middle|\; \|\mathcal{A}\| = n \right\} .$$

For example, let $\tau_s = \langle S^1 \rangle$ be the vocabulary of binary strings, and thus STRUC$[\tau_s]$ is the set of binary strings encoded as logical structures containing a single unary relation.

For any vocabulary τ there is a corresponding first-order language $\mathcal{L}(\tau)$ built up from the symbols of τ and the numeric relation and constant symbols, $=$, $\leq$, BIT, 0, 1, *max*, using logical connectives: $\wedge, \vee, \neg$, variables: $x, y, z, x_1, \ldots$, and quantifiers: $\forall, \exists$.

The numeric relation $\leq$ represents the usual total ordering on the universe $\{0, 1, \ldots, n-1\}$. BIT(x, y) means that when x is coded as a $\log n$ bit number, bit y of this encoding is a one. The relation BIT allows arithmetic operations on $\log(n)$-bit numbers to be first-order definable and it allows subsequences of a $\log(n)$-bit number to be read and copied [Imm99, §1.2]. This will be needed when we simulate a first-order formula φ with the complete formula ζ which may have fewer variables than φ.

We will think of a *static problem*, $S \subseteq$ STRUC$[\tau]$, as a set of structures of some vocabulary τ. It suffices to only consider problems on binary strings, but it is more interesting to be able to talk about other vocabularies, e.g. graph problems, as well. We use the scheme from [Imm99] for coding an input structure as a binary string. If $\mathcal{A} = \langle \{0, 1, \ldots, n-1\}, R_1^{\mathcal{A}}, \ldots, R_t^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}} \rangle$, is a structure of vocabulary τ, then $\mathcal{A}$ will be encoded as a binary string bin$_\tau(\mathcal{A})$ of length $n^{a_1} + \cdots + n^{a_t} + s\lceil \log n \rceil$, consisting of one bit for each a_i-tuple, potentially in the relation R_i plus $\lceil \log n \rceil$ bits to name each constant.[2]

Define the complexity class FO to be the set of all first-order expressible problems. FO is a uniform version of the circuit class AC^0 and it is equal to the set of problems acceptable in constant time on a polynomial size concurrent parallel random access machine [Imm99, Th. 5.22]. When mentioning circuit complexity classes in this paper, we always mean the first-order uniform version of these classes [BIS90].

[2]The details of the encoding are not important, but it is useful to know that for each τ, bin$_\tau$ and its inverse are first-order queries [Imm99, Def. 2.1].

3 Dynamic Problems

We define a *dynamic problem*, $A = \{A_n \subseteq \Sigma_n^\star \mid n = 1, 2, \ldots\}$, to be an infinite sequence of regular languages. For each value of n, A_n is a regular language over Σ_n, a polynomial-size alphabet of operations. If string $w \in \Sigma_n^\star$ is in A_n, we say that the sequence of operations w is *accepted* by the dynamic problem A.

An example of a dynamic problem is the dynamic graph reachability problem DynREACH. This is the dynamic problem induced by the static decision problem REACH. An instance of REACH is a directed graph on n nodes, numbered 0 through $n-1$, with two distinguished nodes s and t. This instance is accepted if there is a directed path from node s to node t. The dynamic problem DynREACH contains, for each n, a regular language DynREACH$_n$ defined over the set of operations

$$\begin{aligned}\Sigma_n \quad \equiv \quad & \{\text{Insert}(i,j), \text{Delete}(i,j) \mid 0 \le i,j < n\} \\ & \cup \{\text{SetS}(i), \text{SetT}(i) \mid 0 \le i < n\} \ .\end{aligned}$$

The operation Insert(i,j) inserts a directed edge from node i to node j in the graph, and the operation Delete(i,j) deletes such an edge. The operations SetS(i) and SetT(i) set or change the start node and the final node of the reachability query. A sequence of operations $w \in \Sigma_n^\star$ is accepted by DynREACH if REACH contains the graph created by sequentially applying the operations in w to the empty graph on n nodes, with s and t initially set to node 0.

4 Dynamic Machines

We next define a model of dynamic computation, leading to a definition of dynamic complexity classes. Our model is based on descriptive complexity [Imm99]; it is suitable for any dynamic computation that maintains a polynomial-size auxiliary data structure.

Definition 4.1 (Dynamic Machine) A *dynamic machine*, $M = \{M_n = (Q_n, \Sigma_n, \delta_n, s_n, F_n) \mid n = 1, 2, \ldots\}$, is a uniform sequence of deterministic finite automata. The dynamic problem accepted by M is the sequence of languages accepted by the DFAs $M_1, M_2, \ldots$:

$$L(M) = L((M_1, M_2, \ldots)) = (L(M_1), L(M_2), \ldots).$$

A state $q \in Q_n$ reached by DFA M_n is the current value of the polynomial-size auxiliary data structure of the dynamic machine. We choose to encode such states as finite logical structures of some vocabulary $\tau = \langle R_1^{a_1}, \ldots, R_t^{a_t}\rangle$[3], i.e. we let

$$Q_n \quad = \quad \text{STRUC}_n[\tau] \ .$$

[3]For simplicity in this paper, we limit τ to be purely relational, with no constant symbols. This is not essential, but it eliminates an extra case in many definitions and constructions.

The alphabets, $\{\Sigma_n \mid n > 0\}$, are described by a finite vocabulary of operator names with fixed arities: $\Sigma = \langle O_1^{r_1}, \ldots, O_s^{r_s}\rangle$. An element of the alphabet Σ_n is an operator name $O_i^{r_i}$ together with an r_i-tuple of parameters drawn from the universe $0, \ldots, n-1$. Omitting the superscripts denoting arity, we have

$$\Sigma_n \ = \ \{O_j(b_1, \ldots, b_{r_j}) \mid 0 \le b_1, \ldots, b_{r_j} < n; 1 \le j \le s\} \ .$$

Thus the size of Σ_n is $O(n^r)$ where $r = \max_j r_j$. Recall that in the example of DynREACH above, we had the operator names Insert, Delete of arity two, and SetS, SetT, of arity one.

The transition functions, $\{\delta_n \mid n > 0\}$, are given by a set of logical formulas $\{\varphi_{i,j} \mid 1 \le i \le t, 1 \le j \le s\}$, defining the new state in terms of the old state and the operation performed. For each relation name, $R_i \in \tau$, and operator name, $O_j \in \Sigma$, the formula $\varphi_{i,j}$ expresses the value of relation R_i in the new state after the transition specified by the operator $O_j(b_1, \ldots, b_{r_j})$. For example, if $\delta_n(q, O_j(b_1, \ldots, b_{r_j})) = q'$, $q = \langle\{0, \ldots, n-1\}, R_1, \ldots, R_k\rangle$, and $q' = \langle\{0, \ldots, n-1\}, R_1', \ldots, R_k'\rangle$, then we have,

$$R_i' \ \equiv \ \varphi_{i,j}(R_1, \ldots, R_k, b_1, \ldots, b_{r_j}, x_1, \ldots, x_{a_i}) \ .$$

If all the formulas $\varphi_{i,j}$ are first order, then the new state, q' is first-order definable from the previous state, q.

The start states, s_n, are given as a set of logical formulas, $\{\sigma_i \mid 1 \le i \le t\}$, giving the initial values of the relations R_i, $1 \le i \le t$. The logic used to express the start states may be different than the logic used to express the transition function, meaning that the initialization of our dynamic machine may have a different computational complexity than the implementation of each operation.

Finally, the final state sets are given by a single sentence α. A state $q \in Q_n$ is in F_n iff $q \models \alpha$. $\square$

Example 4.2 Let PARITY consist of the set of binary strings containing an odd number of "1"s. The problem DynPARITY has unary operators Set and Reset. For a problem of size n, the initial state is a string of n "0"s. Set(i) sets bit i to "1", and Reset(i) sets it to "0". A sequence of such operations is accepted iff the resulting string has an odd number of "1"s.

Define the dynamic machine $P = \{P_1, P_2, \ldots\}$ accepting DynPARITY as follows. The data-structure vocabulary is $\tau = \langle A^1, T^0\rangle$. The relation A will store the current n-bit string, and T will store the current parity of the input. The start state of P is given by the formulas, $A \equiv$ **false**; $T \equiv$ **false**. The final states of P are described by the atomic formula $T()$. The transition functions are described below.

$$\begin{array}{lrcccl} \text{Set}(i): & A'(j) & \equiv & \varphi_{1,1} & \equiv & A(j) \vee i = j \\ & T'() & \equiv & \varphi_{2,1} & \equiv & T() \leftrightarrow A(i) \\ \text{Reset}(i): & A'(j) & \equiv & \varphi_{1,2} & \equiv & A(j) \wedge i \neq j \\ & T'() & \equiv & \varphi_{2,2} & \equiv & T() \leftrightarrow \neg A(i) \end{array}$$

Observe that $L(P)$ = DynPARITY, and each operation in P is first-order definable. In the next section we will define Dyn-FO and it will be obvious that P is a Dyn-FO machine and thus DynPARITY $\in$ Dyn-FO. □

5 Dynamic complexity

Dynamic complexity classes arise naturally from the above definition of a dynamic machine as a family of DFAs. In general, for any static complexity class $\mathcal{C}$ we define the dynamic complexity class Dyn-$\mathcal{C}$ to be the set of dynamic problems that are accepted by dynamic machines whose initial state, transition function, and final set are all computable in $\mathcal{C}$.

If we require that the initial state, transition function, and final set are all described by first-order formulas, we have the dynamic complexity class Dyn-FO. For example, DynPARITY $\in$ Dyn-FO, by the construction in Example 4.2. The class Dyn-FO described here is a slight generalization of the Dyn-FO described by Immerman and Patnaik in [PI97]. The difference is that in [PI97], only the dynamic versions of static problems were considered; here we consider any dynamic problem whatsoever. This treatment provides a more general notion of dynamic computation, which helped us discover the complete problems that we describe in the sequel.

It may be the case that the precomputation of the initial state has greater complexity than the other parts of a dynamic computation. If the computation of δ and F are in $\mathcal{C}$, but the initial state requires complexity greater than $\mathcal{C}$ — up to polynomial — then we say that the dynamic problem is in Dyn-$\mathcal{C}^+$, i.e., Dyn-$\mathcal{C}$ with polynomial precomputation, cf. [PI97].

6 Reductions

Let $A = \{A_n \subseteq \Sigma_n^\star \mid n = 1, 2, \ldots\}$ and $B = \{B_n \subseteq \Gamma_n^\star \mid n = 1, 2, \ldots\}$ be dynamic problems. A simple kind of reduction from A to B maps each operation $\sigma \in \Sigma_n$ to a uniformly bounded sequence of operations $\gamma_1 \cdots \gamma_k \in \Gamma_{r(n)}^\star$ for some polynomial $r(n)$. Note that such a reduction corresponds to a uniform, bounded sequence of homomorphisms from $\Sigma_n^\star$ to $\Gamma_{r(n)}^\star$. Such reductions are related to the bounded reductions investigated in [MSV94] and [PI97].

Definition 6.1 (Bounded Homomorphism)
Let $r(n)$ be a polynomial, let $k \in \mathbf{N}$ be a constant, and let $h = \{h_n \mid n = 1, 2, \ldots\}$ be a sequence of homomorphisms, $h_n : \Sigma_n^\star \to \Gamma_{r(n)}^\star$, such that for all n, all $\sigma \in \Sigma_n$, and all $w \in \Sigma_n^+$, $|h_n(\sigma)| \leq k$ and,

$$w \in A \quad \Leftrightarrow \quad h_n(w) \in B \tag{6.2}$$

Then we say that h is a *bounded homomorphism* from A to B ($h : A \leq_{\text{bh}} B$).

We say that bounded homomorphism h is *uniform* if the map that takes n to h_n has low complexity. For example, if h is uniformly definable by a finite set of first-order formulas — one for each operation symbol $O_j \in \Sigma$ — then f is a bounded first-order (bfo) homomorphism. Not surprisingly, bfo homomorphisms preserve dynamic complexity classes. □

Note that homomorphisms are *memoryless* in that they do not consider previous history when deciding how to map the current operation. It is sometimes useful to consider dynamic reductions that maintain some state as they transform one set of operations to another. However, homomorphisms are simpler and we will use them throughout this paper.

Example 6.3 (Bounded Homomorphism from Dyn-A to Dyn-A') Here is an example of a bounded homomorphism. Given a regular language $A \subseteq \Gamma^\star$, we define two dynamic versions of the membership problem for A. Define the dynamic problem Dyn-$A = \{\text{Dyn-}A_n \mid n = 1, 2, \ldots\}$ such that $w \in$ Dyn-A_n iff,

- The operations are from the set $\{\text{Set}(i, \gamma) \mid 0 \leq i < n,\ \gamma \in \Gamma\}$. $\text{Set}(i, \gamma)$ is interpreted as setting character i of a string to be the symbol γ.
- If we begin with the string $(\gamma_0)^n$ and perform the operations w, then the resulting string is in A.

Define the dynamic problem Dyn-A' as the related problem with operations $\text{Insert}(i, \gamma)$ and $\text{Delete}(i)$, which insert character γ into the string at position i, increasing the length of the string, and delete the character at position i. Again, an operation sequence w is in Dyn-A' iff the sequence, w, applied to the string $(\gamma_0)^n$, yields a string in A. The size parameter n restricts the problem; "Insert" operations that would make the string longer than n characters are ignored.

Given these two dynamic problems, it is easy to see that the operation $\text{Set}(i, \sigma)$ from the first problem can be implemented as the sequence of operations $\text{Delete}(i)$, $\text{Insert}(i, \sigma)$ in the second problem. Thus a bounded homomorphism from Dyn-A to Dyn-A' is given as follows: $r(n) = n$; $k = 2$; and, $h_n(\text{Set}(i, \sigma)) = \text{Delete}(i)\ \text{Insert}(i, \sigma)$ □

We say that a complexity class, $\mathcal{C}$, is *closed under a reduction*, $\leq_r$, iff for all problems A, B, if $B \in \mathcal{C}$ and $A \leq_r B$ then $A \in \mathcal{C}$. It is obvious that almost any conceivable dynamic class is closed under bfo homomorphisms.

Proposition 6.4 *Let $\mathcal{C}$ be any of the complexity classes* AC^i, ThC^i, *$i \geq 0$,* NC^i, *$i \geq 1$,* DSPACE$[s(n)]$, NSPACE$[s(n)]$, *$s(n) \geq \log n$. Then* Dyn-$\mathcal{C}$ *is closed under bfo homomorphisms.*

A difficulty with bfo homomorphisms, is that they only allow a bounded number of operations to set up an initial structure. Thus, a problem that would otherwise be complete might not be if its initial data structure is trivial. For this reason, to build a complete problem for Dyn-FO via bfo homomorphisms, we will have to work hard in §8 to build a universal, first-order formula to place as the main part of the initial data structure of our complete problem.

A different approach would be to increase the power of the reductions so that they can place a polynomially bounded prefix, p_n, in front of the homomorphic image $h_n(w)$ in Equation 6.2.

Thus, define a *bounded homomorphism with polynomial prefix* to be the same as a bounded homomorphism (Definition 6.1) except that there is also a sequence of polynomial-size prefixes: $p(n) \in \Gamma_{r(n)}$, $n = 1, 2, \ldots$, such that for all n, all $\sigma \in \Sigma_n$, and all $w \in \Sigma_n^+$, $|h_n(\sigma)| \leq k$ and $w \in A \;\Leftrightarrow\; p(n)h_n(w) \in B$. Similarly, a bounded first-order homomorphism with prefix (bfop) is a first-order definable bounded homomorphism with polynomial prefix.

It may no longer follow that complexity classes Dyn-$\mathcal{C}$ are closed under bfop's but the corresponding classes with precomputation, i.e. Dyn-$\mathcal{C}^+$ are. In most natural situations, the application of the machine $M_{r(n)}$ for B to the prefix $p(n)$ will result in a uniformly first-order definable data structure for $B_{r(n)}$, and thus the extra precomputation does not take us out of Dyn-$\mathcal{C}$.

Example 6.5 (DynPARITY $\leq_{bfop}$ DynREACH) It is not clear that DynPARITY is reducible to DynREACH via a bfo homomorphism, but it is easy to show that there is such a bfop homomorphism. Namely, let $r(n) = 2n + 2$, and let prefix $p(n)$ construct an initial graph with $s = 0, t = 2n-1$, and the edges $\langle i, i+2\rangle, i < 2n$. This consists of two parallel lines of length n.

The homomorphic image of the operation Set(i) removes the edges $\langle 2i, 2i+2\rangle$ and $\langle 2i+1, 2i+3\rangle$ and inserts the edges, $\langle 2i, 2i+3\rangle$ and $\langle 2i+1, 2i+2\rangle$. Similarly, $h(\text{Unset}(i))$ reverses these operations,

$$\begin{aligned} h(\text{Unset}(i)) \quad \equiv \quad & \text{Insert}(2i, 2i+2), \text{Insert}(2i+1, 2i+3), \\ & \text{Delete}(2i, 2i+3), \text{Delete}(2i+1, 2i+2) \end{aligned}$$

It is easy to verify that this is the required bfop homomorphism. □

7 A Complete Problem for Dyn-FO

In this section we construct a complete problem for Dyn-FO. Let $A = \{A_n \subseteq \Sigma_n^\star \mid n = 1, 2, \ldots\}$ be an arbitrary Dyn-FO problem. Let the operator alphabet be $\Sigma = \langle O_1^{r_1}, \ldots, O_s^{r_s}\rangle$.

Since $A \in$ Dyn-FO, $A = L(M)$ for some machine $M = \{M_n = (Q_n, \Sigma_n, \delta_n, s_n, F_n) \mid n = 1, 2, \ldots\}$, in which δ, s, and F are all first-order definable.

Recall that $Q_n = \text{STRUC}_n[\tau]$ consists of all structures with universe $\{0, 1, \ldots, n-1\}$ for a fixed vocabulary, $\tau = \langle R_1^{a_1}, \ldots, R_t^{a_t}\rangle$.

Let the following first-order formulas be the *definition* of the Dyn-FO machine M: σ_i, $1 \leq i \leq t$, defining the t initial relations; $\varphi_{i,j}$, $1 \leq i \leq t, 1 \leq j \leq s$, defining the new values of each of the t relations in response to each operator, O_j; and α, the acceptance condition.

We next observe that any Dyn-FO machine can be easily transformed to one with trivial initial and accept relations.

Observation 7.1 *Let M be a* Dyn-FO *machine. Then there is an equivalent* Dyn-FO *machine M' such that all the initial relations are empty, i.e, $\sigma_i \equiv$* **false**, *$i = 1, \ldots, t$, and the acceptance condition is also trivial, $\alpha \equiv F$ where F is a new relation of arity 0, i.e., a single additional bit of auxiliary data.*

Proof: Let $\tau' = \tau \cup \{S^0, F^0\}$, i.e., we augment our data structure with two bits: S (start) and F (final). In the initial states, s_n', all relations are empty, i.e., false.

The new transition relations $\varphi_{i,j}'$ are as follows: on input $O_j(b_1, \ldots, b_{r_j})$, do the following,

1. If $\neg S$, apply the initialization definitions, $R_i := \sigma_i$ and $S' :=$ **true**;
2. Apply the transitions as in M, $R_i' := \varphi_{i,j}(b_1, \ldots, b_{r_j})$;
3. Record whether we are in a final state: $F' := \alpha$

Thus M' is exactly equivalent to M and performs the same work that M does, except that at the end of each operation it explicitly records in the bit F whether or not we are currently in a final state. Clearly M' is also Dyn-FO. □

A second observation, which will simplify our construction of a complete problem for Dyn-FO, is that the class of Dyn-FO problems defined by Dyn-FO machines that have a single unary relation as their auxiliary data is complete for Dyn-FO under bfo homomorphisms.

Observation 7.2 *Any* Dyn-FO *problem A is bfo homomorphism reducible to a problem B accepted by a* Dyn-FO *machine whose state space $Q_n = \text{STRUC}_n[\tau_s]$ is the set of all relational data structures with the vocabulary $\tau_s = \langle S^1\rangle$*

consisting of a single unary relation symbol. Furthermore, B's operation vocabulary , $\Sigma = \langle O_1^1, \ldots, O_s^1 \rangle$, consists of only unary operations.

Proof: Let M be a Dyn-FO machine for A, and let the state space of M_n be STRUC$_n[\tau]$, for $\tau = \langle R_1^{a_1}, \ldots, R_t^{a_t} \rangle$. Recall from §2 the first-order transformation bin_τ : STRUC$[\tau] \to$ STRUC$[\tau_s]$, whose inverse is also first-order definable.

We know from [Imm99, Prop. 3.5] that for any first-order transformation I : STRUC$[\alpha] \to$ STRUC$[\beta]$ there is a dual map $\widehat{I} : \mathcal{L}(\beta) \to \mathcal{L}(\alpha)$ such that for all sentences $\theta \in \mathcal{L}(\beta)$ and all structures $\mathcal{A} \in$ STRUC$[\alpha]$,

$$\mathcal{A} \models \widehat{I}(\theta) \qquad \Leftrightarrow \qquad I(\mathcal{A}) \models \theta \ .$$

The first-order formulas defining B are thus given as the image under $\widehat{\text{bin}_\tau^{-1}}$ of the first-order formulas defining A. The homomorphism h is the identity map, and the polynomial blow-up is $r(n) = n^{a_1} + \cdots + n^{a_t}$. We have thus given a bfo homomorphism from A to B. What is going on is that B maintains data structures that are binary strings $\text{bin}_\tau(\mathcal{A})$ exactly simulating the data structure $\mathcal{A}$ of A.

For the last requirement that operations of B all have arity one, we may have to make an additional change. First-order transformations map structures with universe $|\mathcal{A}|$ to structures whose universe is $|\mathcal{A}|^k$, i.e., k-tuples from $|\mathcal{A}|$. Thus, k arguments $a_1, \ldots, a_k$ can be mapped to a single argument $\langle a_1, \ldots, a_k \rangle$. To make the operations of B have arity one it suffices to increase the arity of the first-order transformation bin_τ from A to B so that it is at least as large as the maximum arity of any of the operations of A. □

7.1 Single Step Circuit Value (SSCV)

We now describe the dynamic problem, single step circuit value (SSCV). We will show in Theorem 7.4 that SSCV is complete for Dyn-FO via bfop homomorphims. In Theorem 8.7 we will show that a variant CSSCV of this problem is complete for Dyn-FO via bfo homomorphisms.

SSCV is a not-necessarily-acyclic dynamic circuit value problem. Initially, the problem of size n consists of n "and" gates with current value "0" and no wires connecting them. SSCV allows the following operations:

- Insert(i, j): add a wire from gate i to gate j;
- Delete(i, j): delete any wire from gate i to gate j;
- And(i), Or(i), Not(i): make gate i an "and", "or", "not" gate, respectively;
- Set(i), Reset(i): set the current value of gate i to "1", "0", respectively;
- Step: synchronously propagate values one step through the whole circuit. The new value of an "and" ("or") gate is the conjunction (disjunction) of the current values of the gates connected to its inputs. A "not" gate is actually a "nand" gate; its new value is the negation of the conjunction of its inputs. The value of an "and" gate with no inputs is "1"; the value of an "or" or "not" gate with no inputs is 0.

The acceptance condition for SSCV is that gate 0 has value "1". It is easy to see that SSCV is in Dyn-FO:

Proposition 7.3 *SSCV is in* Dyn-FO.

Proof: We maintain the current value of the circuit as a logical structure with one binary edge relation and four unary relations indicating whether each gate is "and", "or", or "not", and its current binary value.

The operation "Step" is easy to define using first-order formulas.[4] All the other operations simply set one value of the above relations. □

The following is not surprising,

Theorem 7.4 *SSCV is complete for* Dyn-FO *via bfop homomorphisms.*

Proof: Let A be an arbitrary Dyn-FO problem, with corresponding Dyn-FO machine, M, i.e., $A = L(M)$. Combining Observations 7.1 and 7.2 we may assume that M's data structure vocabulary is τ_s, consisting of a single binary string, S, it's initial state is the all-zero string, and its final state is determined by bit 0 of its state string. Furthermore, the vocabulary, $\Sigma = \langle O_1^1, \ldots, O_s^1 \rangle$, consists of s unary operations.

Let $\varphi_1, \ldots, \varphi_s$ be the first-order formulas describing the new value of M's state relation in reaction to operations $O_1, \ldots, O_s$. For an instance of A of size n, there are thus a linear number of possible first-order operations to be performed: $\varphi_i(j)$, $1 \le i \le s$; $0 \le j \le n-1$.

The prefix $p(n)$ of the bfop homomorphism constructs AC^0 circuits for all the operations $\varphi_i(j)$. These are first-order definable [Imm99, Th. 5.22]. Also needed is an n-bit array of latches, to store the current value of M_n's state relation, S. The circuit corresponding to $\varphi_i(j)$, when given control, will start with input S, and over the next depth(φ_i) steps, compute $\varphi_i(j)$, finally returning the relation, $S' \equiv \varphi_i(j, S)$ to the n-bit array of latches when done.

The homomorphic image $h(O_i(j))$ consists of the following depth$(\varphi_i) + 3$ operations to SSCV: set the control bit for $\varphi_i(j)$, Step depth(φ_i) times, latch these new values into the array of latches, unset the control bit.

[4] Note that SSCV is a single iteration of the first-order inductive definition of CVP, cf. [Imm99, Ex. 4.26].

By construction, the prefix and homomorphism are first-order definable, and the resulting SSCV computation exactly simulates M. □

8 A Universal, First-Order Formula, ζ

In order to build a complete problem for Dyn-FO, via bfo homomorphisms, we can no longer include the prefix defining the operations $\varphi_i(j)$ as in Theorem 7.4. We solve this problem, by constructing a first-order formula, ζ, that can simulate any other first-order formula. This section is a diversion from dynamic complexity and may be skipped if the reader so desires.

We now construct a first-order formula, ζ, that is universal in the following sense: any other first-order formula, φ, can be simulated by the formula ζ iterated $d+1$ times, where d is the depth of the parse tree of φ – evaluated on a structure that is a polynomial padding of the original structure, informally,

$$\mathcal{A} \models \varphi \qquad \Leftrightarrow \qquad \mathrm{PAD}(\varphi, \mathcal{A}) \models \zeta^{(d+1)} \tag{8.1}$$

Since ζ has a fixed number of variables it will have to simulate formulas, φ, that have a larger number of variables. A padding of size n^c will let each variable in ζ correspond to $c \log n$ bits. Using BIT we can thus encode c of φ's variables into a single variable of ζ.

Recall that $\tau_S = \langle S^1 \rangle$ is the alphabet of binary strings. The formula $\zeta(y, S) \in \mathcal{L}(\tau_s)$ has one free variable and thus maps binary strings to binary strings, as follows. Let w be a binary string of length n. The structure $\mathcal{A}_w = \langle \{0, \ldots, n-1\}, S_w \rangle$ is an encoding of w. (Note that using the encoding mentioned in §2, the structure $\mathcal{A}_w$ and the binary string $w = \mathrm{bin}_{\tau_s}(\mathcal{A}_w)$ are interchangeable.) Let

$$\zeta(\mathcal{A}_w) \quad = \quad \mathcal{A}_{w'} = \langle \{0, \ldots, n-1\}, S' \rangle \ ;$$

where $S' = \{ i \mid \mathcal{A}_w, i/y \models \zeta \}$.

We may slightly abuse notation and write $\zeta(w) = w'$. For the remainder of this section we fix a vocabulary τ and a formula $\varphi \in \mathcal{L}(\tau)$. We will define the binary string w_φ which will be an encoding of φ and τ that is convenient for the first-order formula, ζ.

Among other things, the string w_φ will represent all the subformulas of φ: $\gamma_1 = \varphi, \gamma_2, \ldots, \gamma_k$. Let r be the number of distinct variables occurring in φ. For simplicity, we will imagine that all r variables occur freely in each subformula γ_i. Let $\sigma = \langle G_1^r, \ldots, G_k^r \rangle$. We will use structures from STRUC$[\sigma]$ to represent the current computation of ζ as it simulates φ and its subformulas.

Suppose now that we are given a structure $\mathcal{A} \in \mathrm{STRUC}_n[\tau]$. Let $\mathcal{B}_0 \in \mathrm{STRUC}_n[\sigma]$ be a blank structure, i.e., all relations are false. Let $\mathcal{C}(\mathcal{A}) = \langle \{0, \ldots, n-1\}, G_1^{\mathcal{C}}, \ldots, G_k^{\mathcal{C}} \rangle$ be the correct structure of φ evaluated on $\mathcal{A}$, i.e., for $i = 1, \ldots, k$,

$$G_i^{\mathcal{C}} \;=\; \{ \langle a_1, \ldots, a_r \rangle \mid \mathcal{A}, a_1/x_1, \ldots, a_r/x_r \models \gamma_i \} \, .$$

The padded encoding of φ and $\mathcal{A}$ – what we called PAD$(\varphi, \mathcal{A})$ in Equation 8.1 – will be the concatenation of three strings: w_φ, $\mathrm{bin}_\tau(\mathcal{A})$, $\mathrm{bin}_\sigma(\mathcal{B}_0)$. The following lemma states the main result of this section, namely that ζ iterated $1 + d = \mathrm{depth}(\varphi)$ times correctly simulates φ.

Lemma 8.2 *There is a universal first-order formula, ζ, such that for any τ, φ, n, $\mathcal{A}$,*

$$\zeta^{(d+1)}(w_\varphi \cdot bin_\tau(\mathcal{A}) \cdot bin_\sigma(\mathcal{B}_0)) \;=\; w_\varphi \cdot bin_\tau(\mathcal{A}) \cdot bin_\sigma(\mathcal{C}(\mathcal{A})) \, .$$

Proof: Although the details of the encoding of w_φ have not been explained yet, it is an easy-for-ζ-to-read representation of φ's alphabet τ and a parse tree for φ. The variables occurring in φ are $x_1, \ldots, x_r$. The subformulas of φ are $\varphi = \gamma_1, \gamma_2, \ldots, \gamma_k$, and the depth of φ's parse tree is d.

We will build ζ so that on input $w_\varphi \cdot \mathrm{bin}_\tau(\mathcal{A}) \cdot \mathrm{bin}_\sigma(\mathcal{B})$, it outputs $w_\varphi \cdot \mathrm{bin}_\tau(\mathcal{A}) \cdot \mathrm{bin}_\sigma(\mathcal{B}')$ where $\mathcal{B}'$ is an "improved approximation" of $\mathcal{C}(\mathcal{A})$. More explicitly, if the input relations $G_i^{\mathcal{B}}$ are correct for all γ_i of height less than j, then the output relations $G_i^{\mathcal{B}'}$ are correct for all γ_i of height less than $j+1$.

Claim 8.3 *Let τ, φ, n, $\mathcal{A}$, k, and σ be as above and let $\mathcal{B} \in$ STRUC$_n[\sigma]$. Then,*

$$\zeta(w_\varphi \cdot bin_\tau(\mathcal{A}) \cdot bin_\sigma(\mathcal{B})) \quad = \quad w_\varphi \cdot bin_\tau(\mathcal{A}) \cdot bin_\sigma(\mathcal{B}') \ ,$$

where for all $j \le d$, if for all γ_i of height less than j, $G_i^{\mathcal{B}} = G_i^{\mathcal{C}(\mathcal{A})}$, then for all γ_i of height less than $j+1$, $G_i^{\mathcal{B}'} = G_i^{\mathcal{C}(\mathcal{A})}$.

Lemma 8.2 will follow immediately from Claim 8.3. Furthermore, Claim 8.3 is a complete specification of the requirements of ζ. On input $w_\varphi \cdot \mathrm{bin}_\tau(\mathcal{A}) \cdot \mathrm{bin}_\sigma(\mathcal{B})$, ζ must do the following:

of ζ

1. Determine n. (Note that the length of ζ's input – and thus the cardinality of ζ's input structure – is $r(n)$ where the polynomial r is determined by φ.)

2. Copy the initial portion of its input, $w_\varphi \cdot \mathrm{bin}_\tau(\mathcal{A})$, directly to the beginning of its output.

3. For each subformula γ_i of height 0, ζ lets $G_i^{\mathcal{B}'} = G_i^{\mathcal{C}(\mathcal{A})}$, i.e., it evaluates the atomic formula γ_i on $\mathcal{A}$ and writes these n^r bits to the correct portion of the output string.

4. For each subformula γ_i of positive height, ζ evaluates γ_i using its definition in w_φ in terms of γ_i's children in φ's parse tree. There are three cases:

(a) $\gamma_i = \neg\gamma_j$: In this case, ζ evaluates $G'_i := \neg G^{\mathcal{B}}_j$.

(b) $\gamma_i = \gamma_j \wedge \gamma_\ell$: ζ evaluates $G'_i := G^{\mathcal{B}}_j \wedge G^{\mathcal{B}}_\ell$.

(c) $\gamma_i = (\forall x_j)\gamma_\ell$: ζ evaluates $G'_i := (\forall x_j < n) G^{\mathcal{B}}_\ell(x_1, \ldots x_r)$.

A key point in the definition of w_φ and the definition of ζ is that ζ be able to compute the positions of each bit of each $G_i(b_1, \ldots, b_r)$, and each input relation, $R(b_1, \ldots, b_a)$. Note that the length of the universe of ζ's input is given by the polynomial,

$$r(n) \quad = \quad |w_\varphi| + |\mathrm{bin}_\tau(\mathcal{A})| + |\mathrm{bin}_\sigma(\mathcal{B})| \; .$$

We will encode the string w_φ to be self-delimiting so ζ can determine its length. $|\mathrm{bin}_\sigma(\mathcal{B})| = kn^r$ since σ consists of k relations each of arity k. For $\tau = \langle R_1^{a_1}, \ldots, R_t^{a_t}, c_1, \ldots, c_s\rangle$, we would have that

$$|\mathrm{bin}_\tau(\mathcal{A})| \quad = \quad n^{a_1} + n^{a_2} + \cdots + n^{a_t} + s\lceil \log n \rceil \quad (8.4)$$

A small problem here is that from ζ's point of view the value of t, the number of terms in this summation, is unbounded. A solution to this problem is to have the convention that the sequence of a_i's in every τ is monotonic. In this case, we can rewrite Equation 8.4 as,

$$|\mathrm{bin}_\tau(\mathcal{A})| \quad = \quad c_1 n^{a_1} + c_2 n^{a_2} + \cdots + c_i^{a_i} + s\lceil \log n \rceil \quad (8.5)$$

where i is the number of different groups of arities. Note that for $n \geq 2$ it must be the case that $i \leq \log|\mathcal{A}|$. This solves our minor problem because summations of size up to $\log n$ are first-order expressible in the presence of BIT [Imm99].

Since $r(n) > n^r$, a single variable of ζ can simultaneously encode values for all r variables of φ.

The full definition of ζ is now straight forward. Recall that $\zeta(S, y)$ takes as input a binary string, $w_\varphi \cdot \mathrm{bin}_\tau(\mathcal{A}) \cdot \mathrm{bin}_\sigma(\mathcal{B})$, encoded via the unary relation, S. y is the free variable of ζ, as ζ computes a unary relation. The output of ζ is the new relation $S'(y) \equiv \zeta(S, y)$. The definition of $\zeta(S, y)$ is as follows.

```
1. compute $n$;    $u = |w_\varphi|$;    $v = |\mathrm{bin}_\tau(\mathcal{A})|$;
2. **if** $(y < u + v)$ **then return**(S(y))
3. **else** compute $i, j$ s.t. $y = u + v + i(n^r) + j$
4. /$\star$ $S(y)$ is $G_i(b_1, \ldots, b_r)$;    $j$ encodes $b_1, \ldots b_r$ $\star$/
5.          compute and return $G'_i(b_1, \ldots, b_r)$
```

In line 5, the new value of $G'_i(b_1, \ldots, b_r)$ is computed according to the instruction in w_φ as in Specification 8.1. This completes the proof of Lemma 8.2. □

8.1 A Universal First-Order Quantifier Block

Here we point out that as a consequence of the existence of the universal formula, ζ, there is a fixed, first-order quantifier block that can simulate any other first-order quantifier block. While not completely surprising, having such a quantifier block seems very valuable and offers insight concerning all the descriptive complexity classes, FO$[t(n)]$.

Recall, [Imm99, Def. 4.24], that for any function $t : \mathbf{N} \to \mathbf{N}$, the descriptive complexity class, FO$[t(n)]$, is the set of static problems $S \subseteq \mathrm{STRUC}[\tau]$ describable by a quantifier-free formula, M_0, a tuple of constant symbols $\bar{c}$, and a quantifier block,

$$\mathrm{QB} \;=\; \left[(Q_1 x_1 . M_1) \ldots (Q_k x_k . M_k)\right] \; ,$$

in the sense that for all $\mathcal{A} \in \mathrm{STRUC}[\tau]$,

$$\mathcal{A} \in S \quad \Leftrightarrow \quad \mathcal{A}, \bar{c}/\bar{x} \models [\mathrm{QB}]^{t(\|\mathcal{A}\|)} M_0 \; .$$

Thus, FO[t(n)] is the set of static problems expressible by first-order quantifier blocks iterated $t(n)$ times. The following equalities are known,

$$\mathrm{AC}^1 = \mathrm{FO}[\log n];\;\; \mathrm{P} = \mathrm{FO}[n^{O(1)}];\;\; \mathrm{PSPACE} = \mathrm{FO}[2^{n^{O(1)}}]$$

A corollary of the existence of the universal first-order formula, ζ, is that there is a complete quantifier block (QBC) in the following sense,

Theorem 8.6 *There exists a fixed quantifier block, QBC, a fixed quantifier-free formula, M_0, and a tuple of constants, $\bar{c}$, such that for any static problem $S \in$ FO$[t(n)]$, there is a corresponding constant, k, and first-order translation, η_S : STRUC$[\sigma] \to$ STRUC$[\tau_s]$, such that for all $\mathcal{A} \in$ STRUC$[\sigma]$,*

$$\mathcal{A} \in S \qquad \Leftrightarrow \qquad \eta_S(\mathcal{A}), \bar{c}/\bar{x} \models [QBC]^{kt(\|\mathcal{A}\|)} M_0 \; .$$

Thus, any block of quantifiers can be linearly simulated by QBC.

8.2 CSSCV: A Dyn-FO-Complete Problem

We now build the problem CSSCV which is the same as SSCV except that the initial circuit includes a first-order definable AC^0 circuit for the universal first-order formula ζ, plus an array of latches and some control logic as in the proof Theorem 7.4. The details of this control logic will be explained in the proof of Theorem 8.7. Call the revised problem, i.e. SSCV plus this first-order definable initial circuit, CSSCV. Obviously CSSCV $\in$ Dyn-FO. We next prove,

Theorem 8.7 *CSSCV is complete for* Dyn-FO *via bfo homomorphisms.*

Proof: We just need to show that every Dyn-FO problem is reducible to CSSCV via a bfo homomorphism. Let $A = L(M)$ be a Dyn-FO problem and machine exactly as in the proof of Theorem 7.4.

An instance of problem A of size n will be mapped to an instance of CSSCV of size $q(n)$ for an appropriate polynomial q. That size $q(n)$ circuit will have an array of latches of length large enough to encode the n-bit binary string, S, together with the encoding w_{φ_i} for $1 \leq i \leq s$, plus kn^r, bits to encode the intermediate-result structure $\mathcal{B}$ as in Lemma 8.2, for any of the φ_i's.

The homomorphic image $h(O_i(j))$ consists of the following operations to CSSCV:

1. Write the finite string w_{φ_i} to the front of the array of latches, shifting the rest of the array to the right.

2. Step $(\text{depth}(\varphi_i) + 1)\text{depth}(\zeta)$ times, thus iterating ζ enough to simulate φ_i. (This results in the binary string $w_\varphi \cdot S \cdot \text{bin}_\sigma(C(S))$ as in Claim 8.3. The beginning of $\text{bin}_\sigma(C(S))$ is an n^2-bit representation of $\varphi_i(S, x, y)$.)

3. Copy row j of $\varphi_i(S, x, y)$ back into the first n bits of the array of latches, zeroing the rest. (Choosing row j corresponds to taking the value just for the desired parameter j.)

Observe that as desired, we have reduced A to CSSCV via a bfo homomorphism. We have only included the main ideas concerning the initial circuits for CSSCV. We have omitted details concerning control lines and copying values, etc. □

9 Other Complete Problems

Now that we know that SSCV is complete for Dyn-FO via bfop homomorphisms, and CSSCV is complete for Dyn-FO via bfo homomorphisms, we can construct related problems that share these properties. First,

Proposition 9.1 *Let* monotone single step circuit value *(MSSCV) be the restriction of SSCV in which there are "and" gates and "or" gates but no "not" gates. Then MSSCV is complete for* Dyn-FO *via bfop homomorphisms, and CMSSCV is complete for* Dyn-FO *via bfo homomorphisms.*

Proof: We use the standard reduction of circuit value to monotone circuit value in which for each gate, g, there is a corresponding gate $\overline{g}$ that computes g's negation. If g is an "and" or "or" gate with inputs $z_1, \ldots z_k$, then $\overline{g}$ is an "or" or "and" gate with inputs $\overline{z_1}, \ldots \overline{z_k}$.

Thus, by at most doubling the size of the corresponding circuits, and the operations that create and modify them, we have transformed any circuit to an equivalent monotone circuit. The results of Theorems 7.4 and 8.7 thus carry over. □

A problem related to MSSCV is the following boolean matrix multiplication problem. An instance of this problem is an $n \times n$ boolean matrix, B, and a boolean vector of length n, X. The operations of this dynamic problem allow the setting or resetting of any bit of the matrix or vector, together with the operation, Step, which replaces the current value of X by multiplying it by the matrix M and then negating it, $X' := \neg MX$. The reason for the negation is that MX computes an n-ary "or" of a binary "and". In two such operations we can compute n-ary "or"s and "and"s.

Proposition 9.2 *The boolean matrix multiplication plus negation problem described above is complete for* Dyn-FO *via bfop homomorphisms. Furthermore, a version of the problem that includes a certain first-order definable initial matrix is complete for* Dyn-FO *via bfo homomorphisms.*

Proof: In two steps of matrix multiplication plus negation, we can simulate all the "or" gates, and then all the "and" gates of MSSCV. □

9.1 Complete Problems For Other Dynamic Complexity Classes

The above complete problems for Dyn-FO can be modified to obtain complete problems for other dynamic complexity classes. Of particular interest is the complexity class Dyn-ThC^0 of those dynamic problems each of whose steps is computable by bounded depth threshold circuits. Analogous to the fact that FO $=$ AC^0, is the result that FO(COUNT) $=$ ThC^0, i.e., ThC^0 is equal to the class of problems expressed by first-order formulas with counting quantifiers [BIS90].

Define the problem SSThCV to be the generalization of SSCV in which threshold gates are used. It is not hard to see that we can generalize the universal formula ζ to a formula ζ_C that is universal for FO(COUNT). Thus, we can also define the problem CSSThCV which is like CSSCV except that we replace ζ by ζ_C. We thus obtain,

Theorem 9.3 *SSThCV is complete for* Dyn-ThC^0 *via bfop homomorphisms, and CSSThCV is complete for* Dyn-ThC^0 *via bfo homomorphisms.*

In a similar, but perhaps less natural way, we can consider other circuit value problems corresponding to the complexity classes, NC^i, AC^i, and ThC^i for $i \geq 1$. In these cases the circuits look like the corresponding NC, AC, or ThC circuits except that they need not be acyclic. The single step operation of SSCV is replaced by a multi-step operation that executes $(\log n)^i$ steps of the circuit. It then follows that the corresponding problems are complete for the complexity classes, Dyn-NC^i, Dyn-AC^i, and Dyn-ThC^i for $i \geq 1$.

10 Conclusions and Future Directions

We have presented the first complete problems for dynamic complexity classes including Dyn-FO and Dyn-ThC^0. We hope and expect that this will enable many other researchers to produce more natural complete problems that will shed further light on dynamic complexity.

We have also produced a universal first-order formula, ζ, and a corresponding complete first-order quantifier block, QBC. We look forward to simpler such universal formulas and quantifier blocks which may help shed light on all the complexity classes FO$[t(n)]$.

The following further directions are promising:

- Now that we have some complete dynamic problems, there is much work to be done classifying many other important dynamic problems; i.e., which are complete, which are reducible to others, etc. It would be nice to know whether dynamic versions of static problems can ever be complete for dynamic complexity classes. (They cannot be complete via memoryless reductions such as homomorphisms. This follows because dynamic versions of static problems are idempotent: if $xy \in A$ then $xyy \in A$ for any sequences of operations x and y. This property is preserved under memoryless reductions, but it is not true of all dynamic problems.)

- It is also very desirable to characterize dynamic complexity from a descriptive point of view. We are beginning this simply by looking at the syntax of the logical formulas that we use to express various dynamic properties. A related, valuable direction is to develop methods to go from a logical description of a dynamic problem, to a good dynamic data structure and algorithm for this problem.

- The complete problems we produced each had two versions: one more natural such as SSCV, and one complete via a weaker reduction such as CSSCV. Further work clarifying the relations between bfo and bfop homomorphisms and other reductions is needed.

References

[BIS90] D.M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *JCSS* 41(3) (1990), 274 - 306.

[DS95] G. Dong and J. Su, “Incremental and Decremental Evaluation of Transitive Closure by First-Order Queries”, *Information and Computation* 120(1) (1995), 101–106.

[Hes01] W. Hesse, “The Dynamic Complexity of Transitive Closure is in TC^0”, *Intl. Conf. on Database Theory*(2001), 234 – 247.

[Imm99] N. Immerman, *Descriptive Complexity*, 1999, Springer Graduate Texts in Computer Science, New York.

[JMS98] H.V. Jagadish, I. Mumick, A. Silberschatz, “View Maintenance Issues for the Chronicle Data Model,” in *Materialized Views: Techniques, Implementations, and Applications*, A. Gupta and I.S. Mumick, eds., 1998, MIT Press, 241 – 253.

[LW99] L. Libkin and L. Wong, “On the Power of Incremental Evaluation in SQL-Like Languages.” DBPL (1999).

[MSV94] P. Miltersen, S. Subramanian, J. Vitter, and R. Tamassia, “Complexity Models for Incremental Computation,” *Theoret. Comp. Sci.* (130:1) (1994), 203–236.

[PI97] S. Patnaik and N. Immerman, “Dyn-FO: A Parallel, Dynamic Complexity Class,” *J. Comput. Sys. Sci.* 55(2) (1997), 199–209.

[Sip83] M. Sipser, “Borel Sets and Circuit Complexity,” *ACM Symp. Theory Of Comput.* (1983), 61–69.