Queries and Algorithms Computable by Polynomial Time Existential Reflective Machines

Extended abstract for PODS'97 submission

Jerzy Tyszkiewicz

Mathematische Grundlagen der Informatik,
RWTH Aachen, Ahornstraße 55,
D-52074 Aachen, Germany.
e-mail `jurek@informatik.rwth-aachen.de`,
phone (+49 241) 80 4557, fax (+49 241) 88 88 215

October 16, 1996

Abstract

We consider two kinds of reflective relational machines: the usual ones, which use first order queries, and existential reflective machines, which use only first order existential queries. We compare these two computation models. We build on already existing results for standard relational machines, obtained by Abiteboul, Papadimitriou and Vianu [APV], so we prove only results for existential machines.

First we show that for both standard and existential reflective machines the set of polynomial time computable Boolean queries consists precisely of all PSPACE computable queries.

The we go farther and compare which classes of algorithms both kinds of machines represent.

It appears that, unless PSPACE = $\mathrm{PTIME}^{\mathrm{NP}}$, there are PSPACE queries which cannot be computed by polynomial time existential reflective machines which use only polynomial amount of relational memory, while it is always possible for standard reflective machines. We conclude that existential reflective machines, although equivalent in expressive power to unrestricted machines, provide substantially worse algorithms than the latter.

On the positive side, we demonstrate that the above quality difference does not affect computing of PTIME queries, what can be done by existential reflective machines in PTIME, even if we take into account query evaluation costs, as well.

Therefore within PTIME we use finer distinctions of the quality of algorithms. We demonstrate that any existential reflective machine computing the transitive closure query must use at least $(1 - o(1)) \log n$ variables, while for reflective relational machines using full first order queries already 3 variables suffice.

This research has been supported by a Polish KBN grant 8 T11C 002 11 and by the German Science Foundation DFG.

1 Introduction

1.1 Class of queries vs. class of algorithms

Many formalisms for representing queries and query languages have been considered and compared with each other. Two most prominent examples are the relational algebra and the predicate calculus, which I prefer to call first order logic. It is one of the early discovered and still most important facts of the database theory that these two formalisms express precisely the same class of queries.

And this is typically the first question one asks when comparing two such formalisms: what are the classes of queries they represent?

However, every formalism comes typically with its own "intended" semantics, and therefore represents not only a class of queries, but a class of algorithms for computing queries, as well. Here the relational algebra and first order logic are not identical any more. Namely for the purpose of query optimization algebra behaves much better. In fact, any practical method of optimization of a query written in first order logic begins with its translation into an equivalent algebra expression, which is then further optimized. So it seems that it is somehow easier to represent efficient algorithms in the algebra.

Thus we see that the issues of representing a class of queries and a class of algorithms, which compute these queries, can be different.

There are even much more drastic examples of this difference. One of them is the result of Suciu and Paredaens [SP] that any algorithm for transitive closure query written in certain complex object algebra must use the powerset operator, and thus requires exponential time and space in the intended semantics, while everybody knows a simple recursive algorithm which does it in quadratic time and quadratic space.

Another example are two kinds of structural recursion of Suciu and Wong [SW], which, if added to the relational algebra for complex objects, represent precisely the same class of queries, but one of them appears to represent much better algorithms than the other does.

1.2 Reflective machines

In this paper we want to discuss yet another example of the mismatch class of queries vs. class of algorithms.

Our starting point is the work of Abiteboul, Papadimitriou and Vianu [APV] about reflective relational machines. These machines are Turing machines which decide classes of structures, communicating with their input structures using first order queries, only; a query, no matter how complicated, is evaluated in one step of the machine. These queries can be dynamically created during the computation—this process is called *reflection*, and hence the name of the machines. Evaluation of a Boolean query provides a feedback to the machine, which can change its behaviour depending on the result; non-Boolean queries update relations in the relational store of the machine.

The authors of [APV] considered these machines as a model of a massively parallel computing device. We believe, however, that it is even a better model of a situation encountered in databases accessed through a network, where users use reflection to circumvent the limitations of the query language—they compute queries not supported in the system by submitting sequences of simpler queries and analyzing their results. This can be achieved either "by hand", or by a specialized software. In this scenario varying the query language used by machines is quite natural, because it reflects the diversity of database systems accessible over networks. Some further argumentation can be found in author's paper [T].

Our two formalisms to be compared are reflective relational machines and a restriction of this model, in which only *semipositive existential* first order queries can be used.

1.3 The results

One of the main results of [APV] is that polynomial time reflective relational machines recognize precisely all PSPACE classes of finite structures, and that this holds even if machines are restricted to be update-free, i.e., to use only Boolean queries.

Our first result is that polynomial time existential reflective machines recognize precisely all PSPACE classes of finite structures, like their unrestricted counterparts do. So the two formalisms represent the same class of queries. The situation changes, if we start considering what classes of algorithms they represent.

We demonstrate that if we impose that polynomial time existential reflective machines can update only relations of bounded degree, then they recognize precisely all $\mathrm{PTIME}^{\mathrm{NP}}$ classes of structures. Thus, unless $\mathrm{PTIME}^{\mathrm{NP}} = \mathrm{PSPACE}$, there are PSPACE classes of, say, graphs which cannot be recognized by polynomial time existential reflective machines which use only polynomial amount of relational store.

Finally we consider what algorithms for PTIME queries our two formalisms represent. If we consider time or space consumption, existential machines are not worse than unrestricted machines, and can implement these queries by PTIME algorithms, even if we count query evaluation costs.

Therefore we use a finer tool for measuring quality of algorithms. We consider the transitive closure query on graphs. This query can be easily computed by a standard relational machine using only 3 variables, essentially by implementing a simple recursive algorithm. However, we demonstrate that any existential reflective machine computing this query must use at least $(1 - o(1)) \log n$ variables in its queries.

1.4 Conclusion

Existential reflective machines compute in polynomial time precisely the same set of queries as their full first order counterparts.

On the other hand, the quality of algorithms for these queries represented by existential machines is worse on very complicated problems, such as those which are (widely believed to exist) in $\mathrm{PSPACE} \setminus \mathrm{PTIME}^{\mathrm{NP}}$, and on some very simple polynomial time queries, as well.

This confirms that we have indeed found a new example of a mismatch between the class of computable queries and the class of algorithms computing these queries, represented in a common formalism.

2 Reflective machines

We define now the *reflective relational machines* in the sense of Abiteboul, Papadimitriou and Vianu [APV], which is in turn an extension of the model of *loosely coupled relational machines* of Abiteboul and Vianu [AV].

Let L be a query language. A *reflective relational L-machine* is defined as follows.

It consists of a standard deterministic Turing machine component, including finite control and several work tapes. In addition it has a *relational store*, consisting of infinitely many relations R_i, $i = 1, 2, \ldots$ of variable arities over some fixed finite set. These relations are provided as an input for the machine. One of the work tapes is distinguished to be the *L-query tape*, on which the machine can write an arbitrary L-query of any finite subsignature of $R_1, R_2, \ldots$, followed by # and a natural number. Upon entering a special state *ex* the query is evaluated in one step on the actual contents of the store, and the result is placed in the relation whose number is the second part of contents of the query tape. In case the second number is 0, the query on the tape should be a sentence. Then the output (*true* or *false*), which is obtained analogously in one step, is used to determine the next state of the machine, along with the symbols seen by heads on the standard tapes.

The initial configuration of the machine is as usually, where the relational store contains the input: it can be an arbitrary finite structure $\mathbb{A}$ of a fixed signature $(R_1, \ldots, R_m)$. The remaining relations $R_{m+1}, R_{m+2}, \ldots$ are set initially empty and will play the rôle of work registers.

We can now easily define, what it means, that a reflective L-machine M computes a total function $f : \mathrm{Fin}(\langle R_1, \ldots, R_m \rangle) \to \mathbb{N}$ or a total query $q : \mathrm{Fin}(R_1, \ldots, R_m) \to \mathrm{Fin}$.

In this paper we consider mainly two query languages: first order logic FO and semipositive existential first order logic EFO, both over the set $\{\neg, \vee, \wedge\}$ of basic Boolean connectives. The first query language is widely known, the latter is a fragment of existential first order logic, in which only input relations $R_1, \ldots, R_m$ can be negated in queries, but they cannot be updated. This applies to the equality relation, as well. All remaining relations can be updated, but must appear unnegated in queries. This implies that EFO is the recursion-free fragment of semipositive Datalog (i.e., Datalog with negation of edb's and inequality).

We call the reflective relational FO-machines simply (standard) reflective machines, and the reflective relational EFO-machines existential reflective machines. In case any of them uses only Boolean queries, we call it *update-free.*

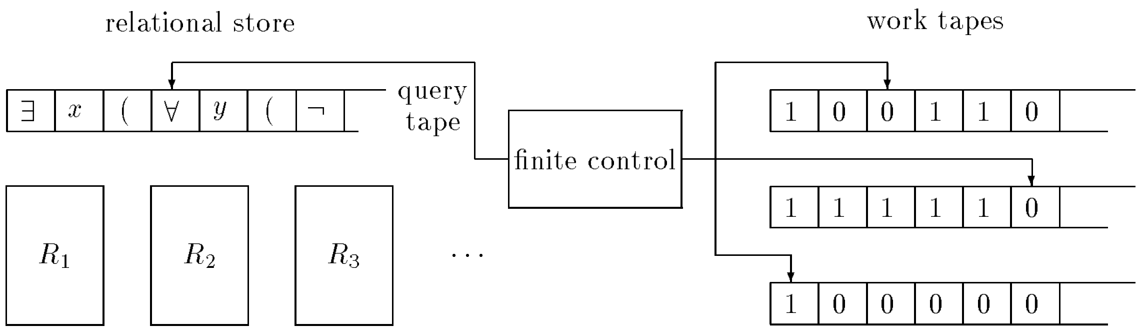

Figure 1: A reflective relational machine.

3 The results

The following paragraphs are devoted to the consecutive levels on which we compare the power of standard reflective machines and of existential reflective machines, as described in the Introduction.

The general reference texts which cover everything what is necessary to read this paper but is not defined here, are books by Abiteboul, Hull and Vianu [AHV] and by Ebbinghaus and Flum [EF]. In fact, each of these books alone already does it.

3.1 Classes of computable queries

First we state the results obtained by Abiteboul, Papadimitriou and Vianu [APV], which tell us about the abilities of standard reflective machines.

Theorem 1. *1. Every* PSPACE *class of finite structures can be decided by an update-free polynomial time reflective machine.*

2. Every polynomial time reflective machine decides a PSPACE *set of structures.* □

Our first result compares the power of existential reflective machines with their standard counterparts in absence of additional restrictions.

Theorem 2. *Polynomial time existential reflective machines can decide precisely all* PSPACE *classes of finite structures.*

Proof. One direction of the theorem follows trivially from Theorem 1, namely an existential reflective machine is a standard reflective machine, as well, so the set of structures it accepts must be in PSPACE.

In the proof of the other direction we want to show that for every polynomial time update-free standard reflective machine there is a polynomial time existential reflective machine, which recognizes the same class of structures. And our only real concern is how to simulate evaluation of an FO sentence using only EFO queries.

We do it as follows: whenever the machine to be simulated attempts to evaluate an FO sentence, the simulating machine rewrites it first to the prenex normal form

$$Q_t x_t \ldots Q_2 x_2\; Q_1 x_1\; \varphi(x_1, x_2, \ldots, x_t),$$

such that all negations in the quantifier-free matrix φ are applied to atoms. Such negations are thus applied to input relations and equalities, only. This rewriting requires renaming variables, if they appear in more than one context in the original formula[1]. In the initial phase our machines goes on asking Boolean queries $\exists x_1 \ldots \exists x_k \bigwedge_{i \neq j} x_i \neq x_j$ for $k = 1, 2, \ldots$, until it finds the first value for which the answer is *false*, determining thereby the cardinality n of its input structure $\mathbb{A}$. Now the simulating machine runs the following algorithm, in which relations R_i of the relational store satisfy $\mathrm{arity}(R_i) = i$. In particular, R_0 is a Boolean flag.

Algorithm A.

> **input** $Q_t x_t \ldots Q_2 x_2\; Q_1 x_1\; \varphi(x_1, x_2, \ldots, x_t)$;
> update R_t with $\varphi(x_1, x_2, \ldots, x_t)$;
> **for** $i = 1$ **to** t
>   **do**
>   **if** $Q_i = \forall$
>   **then** update R_{t-i} with $\exists y_1 \ldots \exists y_n \left(\begin{array}{l} \bigwedge\limits_{j \neq k} y_j \neq y_k \\ \bigwedge\limits_{j=1}^{n} R_{t-i+1}(\frac{y_j}{x_i}, x_{i+1}, \ldots, x_t) \end{array} \right)$
>   **else** update R_{t-i} with $\exists x_i R_{t-i+1}(x_i, x_{i+1}, \ldots, x_t)$
>   **endif**
>   **od**;
> **output** R_0;

The idea behind the algorithm is indeed quite simple. Namely, if we know the cardinality of the structure and can address each element with a separate existentially quantified variable, we can transform equivalently every FO sentence into an existential sentence, replacing universal quantifiers by conjunctions over all elements of the structure. This procedure, however, is easily seen to cause an exponential blow-up of the size of the sentence. To avoid this we save the intermediate results in this process in the relational store and then call them in the next step of the transformation under the name of the store relation. This prevents the syntax from blowing up, causing however blow-up in the relational store, because a relation of arity s requires space n^s to be stored, and we cannot bound s by any constant. As our next result indicates, this cannot be avoided, unless some very unlikely complexity collapse happens. ☐

[1] Note the reversed indexing of quantifiers.

3.2 Algorithms for PSPACE queries

On the face of the problem, polynomial time existential machines and polynomial time standard reflective machines are equivalent, because they compute precisely the same queries. But the situation is more interesting, as the following result holds.

Theorem 3. *1. Every* $\mathrm{PTIME}^{\mathrm{NP}}$ *class of finite structures can be decided by an update-free polynomial time existential reflective machine.*

2. Every polynomial time existential reflective machine which updates only relations of arity $O(1)$ *decides a* $\mathrm{PTIME}^{\mathrm{NP}}$ *set of structures.*

Proof. The proof of Item 1 is presented for directed graphs. It can be easily extended to arbitrary relational structures. We begin it with a purely complexity-theoretic consideration. Let us define that a finite directed graph $\mathbb{G}$ is encoded by a binary string $\mathrm{code}(\mathbb{G})$, resulting from concatenating rows of the adjacency matrix of $\mathbb{G}$. Now let

$$\mathrm{canon}(\mathbb{G}) = \max\{\mathrm{code}(\mathbb{G}') \mid \mathbb{G}' \simeq \mathbb{G}\},$$

where we take max with respect to the lexicographic ordering. $\mathrm{canon}(\mathbb{G})$ is thus a binary string of length n^2, determining the *isomorphism type* of $\mathbb{G}$.

Let L_{canon} be the following language:

$$L_{\mathrm{canon}} = \{\mathrm{code}(\mathbb{G})\#w \mid w \in \{0,1\}^*,\ \exists\, \mathbb{G}' \simeq \mathbb{G}\ \exists v \in \{0,1\}^*\ \mathrm{code}(\mathbb{G}') = w1v\}.$$

It is immediate from the definition that $L_{\mathrm{canon}} \in \mathrm{NP}$. Now consider the following algorithm:

Algorithm B.

> **input** $\mathrm{code}(\mathbb{G})$;
> $w := \epsilon$;
> **for** $i = 0$ **to** $|\mathbb{G}|^2$
>   **do**
>   **if** $\mathrm{code}(\mathbb{G})\#w \in L_{\mathrm{canon}}$
>   **then** $w := w1$
>   **else** $w := w0$
>   **endif**
>   **od**;
> **output** w;

It is easy to see, that this algorithm computes the function $\mathrm{code}(\mathbb{G}) \mapsto \mathrm{canon}(\mathbb{G})$. This confirms the known fact that graph canonization is indeed in $\mathrm{PTIME}^{\mathrm{NP}} = \Delta_2^p$, what has been first proven by Blass and Gurevich [BG].

Now we are turning to the construction of an update-free existential reflective machine, which will decide any given class of structures $\mathcal{C} \in \mathrm{PTIME}^{\mathrm{NP}}$.

Initialization. In the initial phase, exactly as in the previous proof, our machine determines the cardinality n of its input structure $\mathbb{G}$.

Step I. Our next task is to implement Algorithm B on an existential update-free reflective machine, in order to show that the machine we construct will be able to determine $\mathrm{canon}(\mathbb{G})$, where $\mathbb{G}$ is its own relational input. In order to achieve this, it suffices to show how our machine can determine the membership of $\mathrm{code}(\mathbb{G})\#w$ in L_{canon}, for $w = w_1 \dots w_{|w|}$. This is done by the Boolean query

$$\exists x_1 \ldots \exists x_n \left(\begin{array}{l} \bigwedge\limits_{i \neq j} x_i \neq x_j \\ \bigwedge\limits_{\substack{1 \leq i \leq |w| \\ w_i = 1}} R(x_{i \pmod n}, x_{\lfloor i/n \rfloor}) \\ \bigwedge\limits_{\substack{1 \leq i \leq |w| \\ w_i = 0}} \neg R(x_{i \pmod n}, x_{\lfloor i/n \rfloor}) \\ R(x_{|w|+1 \pmod n}, x_{\lfloor (|w|+1)/n \rfloor}) \end{array} \right).$$

The sentence above asserts that there is an ordering of the elements of the universe (namely, according to the indices of variables) such that under this ordering the concatenation of rows of the adjacency matrix of $\mathbb{G}$ agrees with w and the next digit following the copy of w in this concatenation is 1.

Step II. Since the class $\mathcal{C}$ we want to decide is in PTIME$^{\mathrm{NP}}$, there is a polynomial time Turing machine M with an NP oracle, which decides whether $\mathbb{G} \in \mathcal{C}$, given canon($\mathbb{G}$) as input. After two previous steps our machine has already determined canon($\mathbb{G}$). Now it can simulate the behaviour of M in the standard way, provided that we can show how to implement calls to the oracle by using Boolean queries to the relational store.

First of all, the oracle O of M can be w.l.o.g. assumed to consist of binary strings only. Then, according to Fagin's Theorem that NP $= \Sigma^1_1$ [F], there is an existential second order sentence φ such that $w \models \varphi$ iff $w \in O$, where we understand words as structures consisting of the ordering relation $\preceq$ and one unary predicate U, marking 1's in w. Again w.l.o.g. we can assume φ to be of the form $\exists S \vartheta(\preceq, U, S)$, where S is t-ary, and that the negations in ϑ are applied to atoms, only. Moreover, because $\preceq$ is present in the language, we can assume that no equality occurs in ϑ. Let the time complexity of M be bounded by n^k, so the longest query to the oracle it makes is of length at most n^k.

Let $\boldsymbol{x}, \boldsymbol{y}$ denote vectors of variables of length k. Their equality and inequality should be understood vector-like, e.g., $\boldsymbol{x} \neq \boldsymbol{y}$ as $\bigvee_{i=1}^k (\boldsymbol{x})_i \neq (\boldsymbol{y})_i$. Then let $\boldsymbol{s}_i$ be vectors of length t of vectors of variables of length k.

The oracle call of M with word w can be simulated as follows. First the machines checks directly whether or not $\vartheta(\preceq, U, \emptyset)$ holds, and for nonempty S it uses the Boolean query

$$\left(\begin{array}{l} \exists \boldsymbol{y}_1 \ldots \exists \boldsymbol{y}_{|w|} \\ \exists \boldsymbol{s}_1 \ldots \exists \boldsymbol{s}_{|w|^t} \end{array} \right) \left(\begin{array}{c} \bigwedge\limits_{i \neq j} \boldsymbol{y}_i \neq \boldsymbol{y}_j \\ \bigwedge\limits_{i=1}^{|w|^t} \bigwedge\limits_{j=1}^{t} \bigvee\limits_{k=1}^{|w|} (\boldsymbol{s}_i)_j = \boldsymbol{y}_k \end{array} \right) \widetilde{\vartheta},$$

where $\widetilde{\vartheta}$ is obtained from ϑ by replacing every quantifier $\exists y \alpha(y)$ by $\bigvee_{i=1}^{|w|} \alpha(\frac{\boldsymbol{y}_i}{y})$, similarly $\forall y \alpha(y)$ by $\bigwedge_{i=1}^{|w|} \alpha(\frac{\boldsymbol{y}_i}{y})$, and then the atomic formulae $\boldsymbol{y}_i \preceq \boldsymbol{y}_j$ by *true* if $i \leq j$ and *false* otherwise, $U(\boldsymbol{y}_i)$ by *true* if $w_i = 1$ and *false* otherwise; in all these cases we leave negations unchanged, and apply subsequently the obvious simplification procedure, which is doable in PTIME. Finally we replace unnegated atoms $S(\boldsymbol{y}_{i_1}, \ldots, \boldsymbol{y}_{i_t})$ by $\bigvee_{i=1}^{|w|^t} \boldsymbol{s}_i = (\boldsymbol{y}_{i_1}, \ldots, \boldsymbol{y}_{i_t})$ and the negated ones $\neg S(\boldsymbol{y}_{i_1}, \ldots, \boldsymbol{y}_{i_t})$ by $\bigwedge_{i=1}^{|w|^t} \boldsymbol{s}_i \neq (\boldsymbol{y}_{i_1}, \ldots, \boldsymbol{y}_{i_t})$.

The idea behind the above sentence is that elements witnessing $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{|w|}$ constitute the universe of the structure representing w, which is ordered according to the indices of variables. Then any choice of witnesses for $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{|w|^t}$ amounts to choosing a nonempty subset of the t-th cartesian power of the set of witnesses for $\boldsymbol{y}$'s. The way we construct $\widetilde{\vartheta}$ from ϑ reflects precisely this idea.

A fine point here is that ϑ describes the *oracle* and not the query, hence it does not change during the computation, and consequently the process of replacing quantifiers by conjunctions and disjunctions causes only polynomial growth of its size. This finishes the proof of Item 1.

In order to prove Item 2, assume that M is an existential reflective machine, which updates only relations of arity $O(1)$. The task of simulating this machine by an $\mathrm{PTIME}^{\mathrm{NP}}$ machine can be seen as a two-fold task: there is an obvious simulation of what M does on the work tapes, and a task of simulating its relational store actions. Since there is a bound, say k, on the arity of relations it updates, there is no problem with storing them by a polynomial time machine, and updates are not a problem, either, provided we can use an NP oracle to evaluate EFO formulas of polynomial length in polynomial time. The choice of the oracle O is as follows, in which k is a fixed parameter:

$$O = \left\{ \mathrm{code}(\mathbb{A})\#\varphi(x_1,\ldots,x_k)\#n_1\#\ldots\#n_k \;\middle|\; \begin{array}{l} \mathbb{A} \text{ is a finite structure} \\ \varphi(x_1,\ldots,x_k) \text{ is an EFO formula} \\ n_1,\ldots,n_k \text{ are elements of } \mathbb{A} \\ \mathbb{A} \models \varphi[n_1,\ldots,n_k] \end{array} \right\}.$$

O is in NP because in order to verify whether $\mathbb{A} \models \varphi[n_1,\ldots,n_k]$ it is enough to guess witnesses for existential quantifiers in φ, substitute the truth values of atoms according to $\mathrm{code}(\mathbb{A})$ and then evaluate the obtained Boolean expression, which is known to be in PTIME.

Now our oracle is easily seen to be exactly what we needed in order to evaluate EFO formulas, which is in turn enough to perform updates in the simulation. □

There are two nice consequences of our two theorems:

Corollary 4. *Existential polynomial time reflective machines which update relations of arity* $O(1)$ *are able to define all* PSPACE *classes of structures if and only if* $\mathrm{PTIME}^{\mathrm{NP}} = \mathrm{PSPACE}$. □

This indicates that, unless $\mathrm{PTIME}^{\mathrm{NP}} = \mathrm{PSPACE}$, polynomial time existential reflective machines need a relational store of superpolynomial size in order to decide some properties in PSPACE. The conclusion is that existential reflective machines, although they are of the same expressive power as their unrestricted counterparts, implement substantially worse algorithms. This does not depend on the above complexity theoretic assumption, because, despite of many years of attempts, nobody has come up with a $\mathrm{PTIME}^{\mathrm{NP}}$ algorithm for any PSPACE complete problem. This is an evidence, that such algorithms cannot be easy to find or natural. And, apart of this sociological argument, we can even derive some strictly mathematical consequences, which are independent of the above unresolved complexity theory question.

Corollary 5. *There is no uniform simulation of update-free polynomial time standard reflective machines by polynomial time existential reflective machines which update only relations of arity* $O(1)$.

Proof. We use a standard argument from complexity theory. (See a survey of Johnson [J] for more details about using this method, as well as for the reference to the oracle result we mention below.)

Suppose there was such a uniform simulation. Then this simulation would still work even if the machines in question were allowed to use any fixed additional external oracle. This in turn would mean, that in every relativized world $\mathrm{PTIME}^{\mathrm{NP}} = \mathrm{PSPACE}$ would hold, too. But there are known oracles, under which this equality does not hold. This contradiction finishes the proof. □

3.3 Deciding PTIME classes of structures

This is a good news, that the bad behaviour of existential reflective machines does not affect recognition of classes of structures which are in PTIME. Namely, the following holds:

Theorem 6. *Every class of structures which is in* PTIME *can be decided by an update-free polynomial time existential reflective machine. Moreover, it is possible to do it in a way assuring the resulting algorithm to be polynomial time.*

Proof. Principally the following argument is already present in the proof of Theorem 3. First use Algorithm B, implemented on an update-free polynomial time existential reflective machine; then after code($\mathbb{A}$) is computed, the rest of the computation is done entirely on the work tapes.

But this is not quite satisfactory, because Algorithm B is not polynomial time itself (and cannot be even replaced by any polynomial time one, unless PTIME = PTIME$^{\mathrm{NP}}$), so the resulting algorithm, although polynomial time from the point of view of the machine, is not polynomial time, if we count the costs of evaluation of queries, too.

But we can come up with a method, which is polynomial time seen as a whole. It is based on the fact that semipositive Datalog programs capture PTIME over structures with order and constants max, min for its two endpoints [P]. So let P be such a program, defining a class of structures we want to decide. Assume that the sum of arities of all idb symbols in P is k. Now our existential reflective machine first determines, as usually, the cardinality n of the universe. Now it uses precisely one EFO query, which is constructed from P as follows. Let $\exists P^{n^k}$ be the existential closure of the n^k-step unwinding of P. Then the query is

$$\exists x_1 \ldots \exists x_n \bigwedge_{i \neq j} (x_i \neq x_j) \wedge \ \exists P^{n^k} \left(\left\{ \frac{\bigvee_{k \leq l} (y_i = x_k \wedge y_j = x_l)}{y_i \preceq y_j} \right\}_{1 \leq i,j \leq t}, \frac{x_1}{\min}, \frac{x_n}{\max} \right),$$

where $y_1, \ldots, y_t$ are all variables of P and $x_1, \ldots, x_n$ are new variables. The construction corresponds to assuming that the structure is ordered according to indices of x's. M accepts $\mathbb{A}$ if this sentence is true in $\mathbb{A}$ and rejects otherwise.

This is the correct answer, because the final value of P is achieved in an n-element structure in at most n^k iterations. Therefore P and the EFO query we have described above are equivalent in all n element structures.

This implies that, if the query evaluator of the machine satisfies some simple premises, the total complexity (i.e., including the complexity of query evaluation) of the algorithm is within PTIME. □

The conclusion is that if we consider time and relational space usage, algorithms for PTIME queries which can be implemented by both reflective machine types do not differ substantially.

But even despite of the above, existential reflective machines do not behave exactly as good as their unrestricted counterparts within PTIME. In order to demonstrate this, we use a finer measure of the quality of algorithms, namely the number of variables they need. We show now that some apparently very simple polynomial time query can be implemented only by relatively complex algorithms in the model of existential reflective machines.

Theorem 7. *Every existential reflective machines which computes transitive closure of its input graph must use* $(1 - o(1)) \log n$ *variables for inputs of size* n.

Proof. We use here in an essential way a class of graphs $\mathbb{G}_1, \mathbb{G}_2, \ldots$ with the following properties: (i) every satisfiable EFO sentence with k quantifiers is satisfied in $\mathbb{G}_k$ and (ii) $|\mathbb{G}_k| = k2^k$. Such a class has been constructed by Shelah [Sh].

Now let us consider any existential reflective machine M which uses k variables in its computation on $\mathbb{G}_k$. Since the behaviour of the machine depends only on the sentences it evaluates, and, no matter how many times it updates the relational store before evaluating a sentence, this sentence can be expressed in EFO using k variables. If the machine does not halt, then it certainly does not compute the transitive closure of its input. So let us assume it halts and claims that R_s contains the transitive closure of its input $\mathbb{G}_k$. Analyzing this computation of the machine we can reconstruct an EFO formula $\varphi(x, y)$ defining the relation contained in R_s. Let φ have t quantifiers.

Consider a new input graph $\mathbb{G}_k \sqcup \mathbb{H}$ for M, where $\mathbb{H}$ is any graph. Since truth of EFO sentences is preserved under extensions, the new graph satisfies all EFO sentences $\mathbb{G}_k$ does. On the other hand, by (i) $\mathbb{G}_k$ satisfies *all* satisfiable EFO sentences with k variables, hence $\mathbb{G}_k \sqcup \mathbb{H}$ and $\mathbb{G}_k$ satisfy precisely the same EFO sentences with k variables, and in particular the computations of M on $\mathbb{G}_k$ and $\mathbb{G}_k \sqcup \mathbb{H}$ are identical, because the machine is unable to distinguish these two graphs.

First consider $\mathbb{H}$ which is a chain of $t+3$ vertices, and the computation of M on $\mathbb{G}'_k = \mathbb{G}_k \sqcup \mathbb{H}$. Now if $\mathbb{G}'_k \not\models \varphi[a, b]$, where a and b are the two endpoints of $\mathbb{H}$, then M does not compute the transitive closure. So let us assume $\mathbb{G}'_k \models \varphi[a, b]$. Because φ has t quantifiers, there are t elements in $\mathbb{G}'_k$, which witness that $\mathbb{G}'_k \models \varphi[a, b]$. However, $\mathbb{H}$ has $t + 3$ vertices, so there is a vertex c in this chain which is neither a nor b nor any of the witnesses. Let $\mathbb{G}''_k = \mathbb{G}'_k \setminus \{c\}$. Similarly as before, M claims φ defines the transitive closure of $\mathbb{G}''_k$. This is however false, because by the choice of c we have $\mathbb{G}''_k \models \varphi[a, b]$ (all witnesses from $\mathbb{G}'_k$ are in $\mathbb{G}''_k$), while a and b lie in different connected components. This contradiction implies that M cannot compute the transitive closure query.

Therefore any existential reflective machine computing transitive closure must use at least $k + 1$ variables in $\mathbb{G}_k$, and this in virtue of (ii) implies the lower bound of $(1 - o(1)) \log n$ variables. $\square$

It is an easy exercise to construct a standard reflective machine computing transitive closure of graphs, which uses just 3 variables.

3.4 Final remarks

By essentially the same technique we have used proving Theorem 3, one can show the following extension of this result, in which Σ^0_k queries (Π^0_k queries, resp.) are first order queries in prenex normal form, in which there are k blocks of alternating quantifiers, and the first quantifier block is existential (universal, resp.).

Theorem 8. *1. Reflective update-free machines which use Σ^0_k (Π^0_k, resp.) can decide in polynomial time all Δ^p_{k+1} classes of structures.*

2. All classes of structures decided in polynomial time by reflective machines which use Σ^0_k (Π^0_k, resp.) queries and update only relations of arity $O(1)$ are in Δ^p_{k+1}.

Proof. After recalling that $\Delta^p_{k+1} = \mathrm{PTIME}^{\Sigma^p_k} = \mathrm{PTIME}^{\Pi^p_k}$, the proof is quite analogous to that of Theorem 3. All changes are quite straightforward, and the suitable extension of Fagin's result we have used was provided by Stockmeyer [St], who proved that $\Sigma^p_k = \Sigma^1_k$ and $\Pi^p_k = \Pi^1_k$. (The left hand side classes are levels of the polynomial hierarchy, the right hand side classes are sets definable in the prefix classes of second order logic.) $\square$

We are in the position now to extend all the comments we have made comparing algorithms implementable by standard reflective machines and existential reflective machines to the full hierarchy of types of machines, resulting by restricting the quantifier structure of their queries. A version of Corollary 5 can be proven, too, because there are known oracles which separate all levels of the polynomial hierarchy, and the hierarchy from PSPACE, as well. See [J] for details.

Another observation is a kind of polynomial-space-invariance of the expressive power of all the above mentioned kinds of reflective machines. Namely it does not make any difference for them, whether they are allowed to use a polynomial amount of update space or no such space at all.

We should mention that reflective machines which use Σ^0_1 queries are a little bit different from our existential reflective machines. The former can namely negate non-input (i.e., updated) relations from the store, while the latter can not. This difference is important in the proof of Theorem 7, which does not get through for Σ^0_1 machines. The proof of Theorem 2 is however simpler for Σ^0_1 reflective machines.

Finally, there is a well-known heuristics, which says that a good way to create efficient database algorithms is to avoid large intermediate results. It is easily seen that the algorithm we have used in the proof of Theorem 2 violates this heuristics. It follows from Theorem 3 that it must do so, unless $\mathrm{PSPACE} = \mathrm{PTIME}^{\mathrm{NP}}$.

References

[AHV] S. Abiteboul, J. Hull and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[APV] S. Abiteboul, C. Papadimitriou and V. Vianu, The power of reflective relational machines, in: *Proc. 9th Symposium on Logic in Computer Science*, 1994, pp. 230–240.

[AV] S. Abiteboul and V. Vianu, Generic computation and its complexity, in: *Proc. ACM SIGACT Symp. on the Theory of Computing*, 1991, pp. 209–219.

[BG] A. Blass and Y. Gurevich, Equivalence relations, invariants and normal forms, *SIAM Journal on Computing* **123**(1995), pp. 172–184.

[EF] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer Verlag, 1995.

[F] R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in: R. M. Karp, ed., *Complexity of Computation, SIAM-AMS Proc.* **7**(1974), pp. 43–73.

[J] D.S. Johnson, A catalog of complexity classes, in: J. Van Leeuwen (ed.) *Handbook of Theoretical Computer Science, vol. A*, Elsevier, 1990, pp. 67–162.

[P] C. Papadimitriou, A note on the expressive power of prolog, *Bulletin of the EATCS* **26**(1985), pp. 21–23.

[Sh] S. Shelah, Private communication.

[St] L. J. Stockmeyer, The polynomial time hierarchy, *Theoretical Computer Science* **3**(1977), pp. 1–22.

[SP] D. Suciu and J. Paredaens, Any algorithm in the complex object algebra needs exponential space to compute transitive closure, in: *Proc. 13th ACM Symposium on Principles of Database Systems*, 1994, pp. 201–209.

[SW] D. Suciu and L. Wong, On two forms of structural recursion, in: *G. Gottlob, M.Y. Vardi (eds.) Proc. ICDT'95*, Lecture Notes in Computer Science No. 893, Springer Verlag, pp. 111–124.

[T] J. Tyszkiewicz, Fine hierarchies of generic computation, to appear in: *Proc. ICDT'97.*