SERIE B — INFORMATIK

Oracles and First Order Lindström Quantifiers[◇]
A correction to TR # B 94–17

Yachin B. Pnueli*
Janos A. Makowski**

B 94–19
November 1994

Abstract

We investigate the tight relationship between oracles and quantifiers. Our main thesis is that for a logic $\mathcal{L}$ which captures a complexity class $\mathbf{C}$, by enhancing that logic with a Lindström quantifier for some set of structures K, we get a new logic $\mathcal{L}[K]$ which captures $\mathbf{C}^K$ - the class $\mathbf{C}$ relativized to (using an oracle for) the set K. We note however that for this thesis to be correct, we must "equate" the set of *transducers* in both the logic and the complexity class. This brings up a second theme that the relationship between the set of transducers and the set of acceptors (in a logic or complexity class) is less obvious than expected.

◇Part of an on going research work with Janos A. Makowsky.

*Institut für Informatik, Freie Universität Berlin, Takustr. 9, D-14195 Berlin, Germany, e-mail: yachin@inf.fu-berli.de.

**Department of Computer Science, Technio-Israel Institute of Technology, Haifa, Israel, E-mail: janos@cs.technion.ac.il

1 Introduction

Throughout our talk, we consider only objects suitable for both computers and logic i.e. vocabularies are finite, structures are both finite and ordered and sets of structures are closed under isomorphisms. Parts of the presentation can be applied to other cases, but this does not concern us.

Furthermore we assume the reader is familiar with the basic methods and results of both finite model theory and complexity theory. In the more formal part of this presentation we shall give explicit definitions of all but the most standard terms we use, but in the introduction and after thought sections we shall often use terms and results without explicitly defining them.

The classical results of finite model theory [Bü60, Fag74, Imm87, Sto87, AV88] showed that there is a tight relationship between logics and complexity classes. Namely for most "natural" logics and complexity classes there are complexity class which captures a given logic and a logic (or sub logic) which captures a given complexity class.

Büchi - Monadic SOL on strings captures the REGULAR LANGUAGES.
Fagin - Existential SOL captures **NP**.
Stockmeyer - Each level of the Polynomial Hierarchy **PH** is captured by some level of the fixed alternation hierarchy of SOL.
Immerman - $FOL[DTC], FOL[TC], FOL[ATC]$ capture $\mathbf{L}, \mathbf{NL}, bP$.
Abiteboul/Vianu - $LFP = IFP$ captures **P** and PFP captures **PSpace**.

Recent results by [MP93, Ste93b]

MP93 - $FOL[HEX]$ captures **PSpace**.
Stewart $FOL[HAM]$ captures $\mathbf{L}^{\mathbf{NP}}$.

while in the same spirit as the above, indicated that in some cases this relationship is very "modular". This modularity was made explicit in [MP94] where it was specifically stated:

> For Immerman's logics $FOL[DTC], FOL[TC], FOL[ATC]$ further enhancement of the logic by a Lindström quantifier for a set K gives a new logic $FOL[DTC, K]$ etc. which captures the original class relativized to K - for example $\mathbf{L}^K$.

Two remarks should be immediately made with respect to this result:

(i) To make these results work one has to use a very precise model of log space oracle computations.
(ii) In the original paper [MP94] it was WRONGLY claimed that $FOL[ATC, K]$ captures $\mathbf{P}^K$. The complexity class captured is only $\mathbf{AL}^K$ with a specific oracle computation model for **AL** in which $\mathbf{AL}^K \neq \mathbf{P}^K$ for arbitrary K.

The important point of these "modularity" results was not that yet a few more complexity classes (or logics) where captured, but rather that the capturing is achieved independently of the semantics of K i.e. if M is some oracle using machine and ϕ is some formula with a Q_K quantifier, such that for a given set K the set defined by ϕ equals the set recognized by M, then if we connect M to an oracle for a different set K' and change the semantics of the Q quantifier to be that of K', the "new" formula defines exactly the set recognized by the "new" machine.

The main thesis of this talk is that this modular relationship is not limited to the specific logics Immerman used but rather is true for arbitrary logics. Specifically we show that by enhancing various fragments of SOL with a Lindström quantifier Q_K we capture the various relativized versions of the Σ and Π classes of **PH**, and this capturing is independent of the semantics of K.

One may now take any logic $\mathcal{L}$ which captures some class **C**, and ask if the class captured by $\mathcal{L}[K]$ is $\mathbf{C}^K$? Naturally the answer depends on what we mean by $\mathbf{C}^K$. It turns out that in many cases, specifically when $\mathcal{L}$ itself already contains a Lindström quantifier as part of its syntax, the captured class is different from what is usually understood by $\mathbf{C}^K$. Apparently such results contradict our above thesis.

To understand why we get such results and why we believe they do not contradict our "thesis" we look a bit more closely into what constitutes a complexity class.

One usual way to define complexity classes is to consider such a class, say **C** as the set of all computation devices (Turing machine) which obey certain restrictions on resources such as time, space, etc. Under such a definition a complexity class naturally includes various "types" of machines - acceptors, transducers and Turing transducers.

Definition 1.
- *Acceptors* are machines which given an input structure either accept or reject it (typically by entering one of two distinguished states). An acceptor can *recognize* a set K - i.e. accept all members and reject all non-members of K.
- *Transducers* are machines which given an input structure produce some other structure as output. A transducer can *reduce* a problem (set of σ–structures) K_1 to a problem (set of τ–structures) K_2 by producing for each σ–structure $\mathcal{A}$ a τ structure $\mathcal{B}$ such that $\mathcal{A} \in K_1$ iff $\mathcal{B} \in K_2$.
- *Turing Transducers* or *relativized acceptors* are acceptors which have access to an oracle, such that given an input structure they either accept or reject (typically via consulting the oracle some times during their operation). A Turing transducer *Turing reduces* K_1 to K_2 if when it is connected to an oracle for K_2 it recognizes the set K_1.

We remark that while it is possible to make the distinction between the three "types" of machines quite fuzzy, in context they have a clear meaning. For example if HAM is the set of graphs having a Hamiltonian path, the statement "$HAM \in \mathbf{NP}$" clearly speaks about **NP**–acceptors. The statement "HAM is **NP**–complete to many one **P** reductions" clearly speaks about **P**–transducers (and **NP**–acceptors) and a question such as "Does the Polynomial Hierarchy collapse ?" speaks about various Turing transducers.

Assume we have some complexity class **C** which is defined to include exactly all acceptors obeying certain resource bounds. Naively one might assume that determining which transducers and Turing transducers should "naturally" be included in such a class is a trivial matter. Indeed these issues where first raised for the class **P** where there is universal agreement on this matter and any "natural" suggestion is equivalent to any other.

The problem becomes less obvious when more "difficult" classes are investigated. For example, the class **L** has at least 4 plausible models of how to define the set of LogSpace bounded Turing transducers. (Simon's bounded model, Ladner Lynch's unbounded model, Wilson and Buss stack model and Lynch and Buss's multi tape model, not to mention a fifth variation suggested by the authors of this work in [MP94] - see references therein for the other models).

The second main thesis of this talk is that this difficulty, in defining for a given class of acceptors, its "natural" sets of transducers and Turing transducers is inherent, in the sense that, looking at complexity classes from the viewpoint of finite model theory, there is no "unique choice".

Combining the two lines of thought, if we now understand **C** to be a class containing a given set of acceptors, a given set of transducers and a given set of Turing transducers, then if the whole of **C** is captured by some logic **L** (with or without Lindström quantifiers), then $\mathbf{C}^K$ will be captured by $\mathcal{L}[K]$, and this in a way independent of the semantics of K.

Furthermore if **C** and **D** are two "whole" complexity classes captured by some logics $\mathcal{L}_{\mathbf{C}}$ and $\mathcal{L}_{\mathbf{D}}$ both based on the same "pure" logic, then if there is an oracle such that $\mathbf{C}^K \neq \mathbf{D}^K$ this implies $\mathbf{C} \neq \mathbf{D}$. This however is often a very undramatic result, as the distinction might lie in the classes of transducers or Turing transducers and not in the class of acceptors which is usually the one of interest.

For example the Baker, Gill and Solovey oracle K such that $\mathbf{P}^K \neq \mathbf{NP}^K$ does demonstrate that our "whole" class **P** is different from the "whole" class **NP** since it proves that the class of **P** Turing transducers is distinctly weaker than the class of **NP** Turing transducers. But this is a long way from answering whether the class of **P** acceptors differs from the class of **NP** acceptors - which is the "real" **P**?**NP** question.

As we claim that from the finite model theoretic point of view the relationship between classes of acceptors and classes of transducers is "weak", such separation results teach us very little (if anything) about those complexity theoretic results which are of interest to us.

Finally we admit that this presentation has a third aim, which is to correct mistakes in the published version of [MP94]. We shall therefore review the results of that publication in some detail and point out the errors and corrections where appropriate.

2 Enhancing a Logic by a Lindström Quantifier

The aim of this section is to define $\mathcal{L}[K]$ and as we go through the definition to show the "logical" analogons of acceptors, transducers, oracles, and Turing transducers. As we assume the audience to be familiar with the notion of a logic enhanced by a Lindström quantifier we allow ourselves to speak about such logics before they are defined. To completely formalize the presentation one should apply the definitions first only to "pure logics" (without Lindström quantifiers) and then build the other cases inductively.

We remark that most of our definitions are in themselves not new and can be found already in [Ebb85] of [BF85]. The only difference is that we include vectorization inside our notion of feasibility (as was done in [MP94]) and that we allow free second order variables where these are meaningful.

Let $\mathcal{L}$ be a logic or a fragment of a logic. As a guiding example consider $\mathcal{L}$ as either FOL or SOL, however the definitions apply inductively so that $\mathcal{L}$ can already be a logic enhanced with some Lindström quantifiers. Also we believe that our definitions apply to fixed point logics such as LFP, IFP and PFP logics as well, but have not yet verified our definitions with respect to these logics.

The first analogy we draw is that a sentence (or a formula without free variables) is the analog of an acceptor. $\phi = M$ iff each τ–structure is either accepted by M and satisfies ϕ or is rejected by M and does not satisfy ϕ.

Clearly the set of all sentences of a logic is then the analog of the set of all acceptors in a complexity class closed under various basic operations. We observe that the usual notion of *capturing* is *"for every formula defining some set S there is an acceptor for S and for every acceptor of some set S there is a formula defining S"*. As this notion speaks only about acceptors and sentences of the logic, the "classical" results of finite model theory about logics capturing complexity classes, as they are usually stated, are too statements only about classes of acceptors.

One may now ask what is the analog of a formula with free variables ? While for such an object we do not have a clear "complexity theoretic counterpart" if we group some such formulas together we get the counter part of a transducer. This is done in two steps as follows:

Definition2 (Feasibility). Let $\tau = \{R_1, \ldots, R_m\}$ be some vocabulary. For R_i a relation, let $\rho(R_i)$ denote its arity. Let $\Phi = \langle \phi, \psi_1, \ldots, \psi_m \rangle$ be formulas of $\mathcal{L}$ over a (possibly different) vocabulary σ. We say that Φ is *k–feasible for τ over σ* if the following hold:

(i) ϕ has k distinguished distinct (first order) free variables.
(ii) Each ψ_i has $k\rho(R_i)$ distinguished distinct (first order) free variables.
(iii) It is possible for Φ to have other non–distinguished free (first and second order) variables. These do not have to be distinct among the component formulas of Φ.

Φ is *feasible for τ over σ* if it is k–feasible for τ over σ for some integer k.

Observe that if Φ is feasible for τ over σ, then it can be regarded as a *logical reduction* transforming a σ structure into a τ structure. We formalize this notion as follows:

Definition3 (The structure $\mathcal{A}_\Phi$). Let $\mathcal{A}$ (with universe A) be a σ–structure and Φ be k–feasible for τ over σ. Given a substitution $\mathcal{Z}$ for all none distinguished free variables of Φ (FO variables to elements of A and SO variables to appropriate subsets of A), the structure $\mathcal{A}_\Phi$ is defined as follows:

(i) The universe of $\mathcal{A}_\Phi$ is the set $A_\Phi = \{\bar{a} \in A^k : \mathcal{A}, \mathcal{Z} \models \phi(\bar{a})\}$;
(ii) The interpretation of R_i in $\mathcal{A}_\Phi$ is the set

$$\mathcal{A}_\Phi(R_i) = \{\bar{a} \in {A_\Phi}^{\rho(R_i)\cdot k} : \mathcal{A}, \mathcal{Z} \models \psi_i(\bar{a})\};$$

Note that $\mathcal{A}_\Phi$ is a τ–structure of cardinality at most $\mid A \mid^k$, hence our analogy works only for poly-size transducers.

Having the analog of a reduction or transducer, we naturally proceed to suggest the following as the analog of an oracle query:

Definition 4 (The Q_K Formation Rule). Let K be a set of τ–structures closed under isomorphism and Φ be a set of σ–formulas of the logic $\mathcal{L}$. The Q_K formation rule, states that

$$Q_K\Phi$$

is a formula where all the distinguished free variables of Φ are bound. Given a substitution $\mathcal{Z}$ to the remaining free variables of Φ, $\mathcal{A} \models Q_K\Phi$ iff $\mathcal{A}_\Phi \in K$.

Observe that as with the complexity theoretic notion of an oracle query (or consultation) this notion is independent of the semantics of K.

We conclude this part of the "story" by suggesting that the analog of the set of Turing transducers is the set of sentences of $\mathcal{L}[K]$ as follows:

Definition 5 (The Logic. $\mathcal{L}[K]$) The enhancement of a logic $\mathcal{L}$ by a quantifier for K ($\mathcal{L}[K]$) is obtained by adding the Q_K formation rule to the set of formation rules of $\mathcal{L}$ and by adding the interpretation of this rule to the set of interpretation rules of $\mathcal{L}$.

3 The Power of $\mathcal{L}[K]$ - The First Order Case

We now review, in light of our above analogies, some old and new results about logics of the form $\mathcal{L}[K]$ where $\mathcal{L}$ is FOL possibly enhanced by some set of Lindström quantifiers. We start by restating the first such results due to Immerman [Imm87]:

$$FOL[DTC] = \mathbf{L} \qquad FOL[TC] = \mathbf{NL} \qquad FOL[ATC] = \mathbf{AL} = \mathbf{P}$$

We remark that while the first three of these results are obtained in a direct way by mapping the set of all configurations of log space bounded machines to the set of k–tuples of the input structure, $FOL[ATC] = \mathbf{P}$ was obtained indirectly from $FOL[ATC] = \mathbf{AL}$ by a theorem of Chandra, Kozen and Stockmeyer [CKS] equating the sets of **AL** acceptors and **P** acceptors.

One could think of an alternative proof:

(i) Show that the set of **L** transducers is captured by the set of all Φ vectors of formulas over $FOL[DTC]$ (say under the Ladner Lynch unbounded model [LL76]). This is easy since one can map the set of all configurations of a log space bounded machine to k–tuples of the input structure.
(ii) As ATC is a problem **P** complete for (many one) **L** reductions, each acceptor in **P** has an equivalent formula of the form $ATC\Phi$ where Φ is in $FOL[DTC]$.
(iii) As $FOL[ATC, DTC]$ can be model checked in **P**, this logic captures **P**.
(iv) As DTC is definable in $FOL[ATC]$, $\mathbf{P} = FOL[ATC, DTC] = FOL[ATC]$.

In fact a similar proof was used in [MP93] to show that $FOL[HEX] = \mathbf{PSpace}$, where HEX is the set of graphs over which player one (the connector) has a winning strategy in the game of Hex as defined in [GJ79].

(i) The sets of **L**, **NL** and $\mathbf{AL} = \mathbf{P}$ transducers are captured by the sets of vectors of formulas Φ in $FOL[DTC]$, $FOL[TC]$ and $FOL[ATC]$ respectively.

(ii) HEX is **PSpace**–complete for (many one) **L** reductions, so each **PSpace** acceptor is equivalent to a formula $HEX\Phi$ where $\Phi \in FOL[DTC]$.
(iii) $FOL[HEX, DTC]$ can be model checked in **PSpace**, so this logic captures **PSpace**.
(iv) DTC is definable in $FOL[HEX]$ so **PSpace** $= FOL[HEX]$.

Similar claims where made about $FOL[ATC, A] = \Delta_i^P$ where A is some problem Δ_i^P–complete for many one polynomial reductions (see [Joh90] for examples of such problems).

Remark. 1. To prove that the set of **AL** and **P** transducers coincide we use the equality of the sets of acceptors (consider the language L where the input is the set of pairs (structure,number) such that the bit marked by the number in the output of the transducer is a 1.

2. Given that the notion of a **L** reduction is fixed, there are two possible notions of what is an **NL** reduction is (the usual one and the γ–reduction one) it can be shown (see appendix A) that they coincide in terms of computational power.

The fact that our results about transducers could be generalized to Turing transducers was first hinted in a paper by Stewart who showed [Ste93a,Ste93b] that $FOL[HAM] = \mathbf{L}^{\mathbf{NP}}$. We note that while he did not state this explicitly, and while he did rely on the exact semantics of HAM for his result, the very complexity class he captured indicates this.

The generalization to Turing transducers was first noted (albeit with errors) in [MP94]. The correct version of these results is:

Theorem 6. *For arbitrary K the logics $FOL[DTC, K]$, $FOL[TC, K]$ and $FOL[ATC, K]$ capture the classes $\mathbf{L}^K$, $\mathbf{NL}^K$, and $\mathbf{AL}^K$ respectively.*

So that we have that the logics $FOL[DTC]$, $FOL[TC]$ and $FOL[ATC]$ completely capture the classes **L**, **NL** and **AL**.

Remark. 1. For all the above classes there is no one agreed upon set of "natural" Turing transducers. Indeed to make our results hold, a **very specific** set of such Turing transducers has to be chosen - the so called unbounded fixed stack model. For the **L** case the oracle computation model (set of Turing transducers) outlined in [MP94] is correct as are the proofs for that case. For the **NL** case the model as presented in [MP94] is too weak and the result does not follow, but some rather minor technical corrections can be made and in the new model the results are correct and the proof is given in appendix B. For the **AL** case again a correction has to be made and the proofs are given in the appendix, but here also we must explicitly state an error in [MP94] in that WE DO NOT CAPTURE THE CLASS $\mathbf{P}^K$.

One can define the set of Turing transducers of **AL** in more than one way, and specifically there are two very close models which differ only in the way they restrict "non-deterministic alternations" such that in one case

$$FOL[ATC, K] = \mathbf{AL}^K \text{ but for some } K\text{: } \mathbf{AL}^K \subset \mathbf{P}^K$$

and in the other

$$FOL[ATC, K] \subset \mathbf{AL}^K \text{ for some } K\text{, but: } \mathbf{AL}^K = \mathbf{P}^K$$

One may wonder whether other $\mathcal{L}[K]$ first order based logics such as $FOL[HAM, K]$ or $FOL[HEX, K]$ capture interesting classes such as $\mathbf{NP}^K$ or $\mathbf{PSpace}^K$. Also one might question whether the above "gap" and our inability to capture $\mathbf{P}^K$ is not a mere coincidence due to the incompetence of the authors in defining just the right Turing transducer set which achieves both aims.

We claim this is not the case, and that the answer to the above is negative. Indeed more specifically we claim that the set of Turing transducers of a logic of the form $FOL[A_1, A_2 \ldots A_m]$ is in a strong sense bound to the set of logspace bounded transducers as the following theorem shows:

Theorem 7. *For any logic $\mathcal{L}$ of the form $FOL[A_1, \ldots A_m]$ there is a set of structures K such that $\mathbf{P}^K \not\subseteq \mathcal{L}[K]$ (indeed $\mathbf{NC}_2^K \not\subseteq \mathcal{L}[K]$).*

Proof: See appendix C □

Note that this is a very different result than what we can obtain for many one transducers:

Theorem 8. *If $\mathcal{L}$ captures $\mathbf{C}$ in the usual (acceptor) sense, than any $\mathbf{C}$ many one reduction which is both functional and polynomial can be captured by a $\Phi \in \mathcal{L}$.*

[: Proof outline] Let M be a $\mathbf{C}$ transducer. If it is functional we can map each input structure to a unique output structure and given an agreed upon encoding scheme to a unique string. Given that such a "function" exists we can define a language L over pairs of a structure and a number such that $(\mathcal{A}, n) \in L$ if on input $\mathcal{A}$ the n'bit of the output is a 1. If $M \in \mathbf{C}$ ($\mathbf{L} \subseteq \mathbf{C}$) then the acceptor of L is also in $\mathbf{C}$ and we have a formula over a dictionary which extends that of the input structure by some constants needed to represent n. If the reduction is polynomial we can replace them but a tuple of free variables over the input structure and we get the reduction formula. □

We conclude this section by investigating the relationships between classes $\mathbf{C}$, $\mathbf{D}$, $\mathbf{C}^K$ and $\mathbf{D}^K$ where $\mathbf{C} \subseteq ?\mathbf{D}$. Specifically returning now to Stewart's result, theorem 6 shows that

$$FOL[HAM] = FOL[DTC, HAM] = FOL[TC, HAM] = FOL[ATC, HAM]$$

and hence that

$$\mathbf{L}^{\mathbf{NP}} = \mathbf{NL}^{\mathbf{NP}} = \mathbf{AL}^{\mathbf{NP}}$$

. (Where $\mathbf{AL}^K$ uses the weaker model). Actually this is true for any quantifier which can express all three reachability relations (such as HEX).

In a more general setup we can show

Theorem 9 (ABC). *Let A, B, C be three sets of structures such that $FOL[A, C] \not\subseteq FOL[B, C]$ then $FOL[A] \not\subseteq FOL[B]$*

Proof: See appendix D. □

As a corollary we get that

Corollary 10. *Let $\mathbf{D}$ and $\mathbf{C}$ be two of $\mathbf{L}$, $\mathbf{NL}$, $\mathbf{AL}$, then if there is an oracle K such that $\mathbf{D}^K \neq \mathbf{C}^K$ (under the appropriate transducer models) then $\mathbf{D} \neq \mathbf{C}$.*

For the $\mathbf{L}$ vs. $\mathbf{NL}$ case this result was first claimed by Wilson [Wil86] for his slightly different model, for a model where the oracle tape is within the space bound it was proven even earlier.

A second corollary is

Corollary 11. *If there is a problem $\mathbf{NP}$–complete ($\mathbf{PSpace}$–complete) for many one $\mathbf{P}$ reductions, which is provably not $\mathbf{NP}$–complete ($\mathbf{PSpace}$–complete) for many one $\mathbf{L}$ reductions then $\mathbf{L} \neq \mathbf{P}$.*

Note that if we had that $\mathbf{P}^K = FOL[ATC, K]$, theorem 9, together with the oracles of Buss [Bus88] or Wilson [Wil86] separating $\mathbf{NL}^K$ from $\mathbf{P}^K$ would imply $\mathbf{NL} \neq \mathbf{P}$!

Similarly for $FOL[HAM]$ and $FOL[HEX]$ and the appropriate oracles separating $\mathbf{NP}$ and $\mathbf{PSpace}$ from lower classes.

4 The Power of $\mathcal{L}[K]$ - Beyond the First Order Case

In the literature one finds many cases of classes such that $\mathbf{D} =? \mathbf{C}$ is a long standing open problem while there are oracles K such that $\mathbf{D}^K \neq \mathbf{C}^K$. Besides the above mentioned $\mathbf{L}^K \neq \mathbf{P}^K$ example, the most important examples are the results of Baker, Gill and Solovey ([BGS75]) - $\mathbf{P}^K \neq \mathbf{NP}^K$, and further works along this line (see Ko [Ko89] and references therein) which show oracles which separate the polynomial hierarchy at any desired level and also from $\mathbf{P}$ and $\mathbf{PSpace}$ (where $(\Sigma^P_{n+1})^K$ is defined as $\mathbf{NP}^{(\Sigma^P_n)^K}$).

Our theorem 9 gives hope that if one succeeds in capturing a complexity class in "whole", one might be able to utilize results about the relativized class to learn more about the unrelativized class. One the other hand our theorem 7 shows that for most interesting cases enhancements of FOL are not powerful enough, we hence focus our attention on logics with a more powerful syntax, specifically on SOL.

Well known results by Fagin, Meyer and Stockmeyer [Fag74, MS72, Sto87, GJ79] show that each level of the polynomial hierarchy ($\mathbf{PH}$) is captured by some fragment of SOL and that the entire $\mathbf{PH}$ is captured by SOL. Our results here are that for each such fragment of SOL which captures a level of the polynomial hierarchy, enhancing that fragment with a quantifier for K captures that level of the polynomial hierarchy relativized to an oracle for K. Furthermore, the entire polynomial hierarchy relativized to K is captured by $SOL[K]$. As a by product of our proof method we obtain (non constructively) that every formula of $SOL[K]$ has an equivalent in prenex normal form.

We note that as $\mathbf{NP}^K$ is captured by an enhancement of existential SOL with some quantifier and not by an enhancement of FOL, one cannot directly deduce from $\mathbf{P}^K \neq \mathbf{NP}^K$ results much about the notorious $\mathbf{P}vs.\mathbf{NP}$ question, in fact the conclusions to be drawn are of an opposite flavor: It is possible to have two logics $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1 \subseteq \mathcal{L}_2$ while $\mathcal{L}_1[K] \not\subseteq \mathcal{L}_2[K]$ provided the *base syntax* is different - viewed in complexity theoretic terms, this means that one can consistently define two complexity classes which contain the same set of acceptors but different sets of transducers, while retaining the natural closure properties one expects of a complexity class.

5 Sub-Logics of $SOL[K]$ and Proofs

Definition 12. (Sub logics of $SOL[K]$)

(i) $INSOL[K]$ is the restriction of $SOL[K]$ to formulas where all second order quantifiers appear outside any Q_K quantifier. (The closure of $FOL[K]$ to the formation rules of SOL).
(ii) $NFSOL[K]$ (normalized extended SOL) is the restriction of $SOL[K]$ to formulas where all second order quantifiers appear outside any other elements of the formula.
(iii) $\exists SOL[K]$ ($\forall SOL[K]$) - existential (universal) extended SOL - is the restriction of $SOL[K]$ to formulas $\exists R\phi$ ($\forall R\phi$) where ϕ does not contain any second order quantifiers.
(iv) In general, for X a string of $\forall$'s and $\exists$'s, we denote by $XSOL[K]$ the restriction of $NFSOL[K]$ to formulas where the (alternating) quantifiers form a sub pattern of X. For example $\exists\forall\exists\forall SOL[K]$ can contain formulas of three alternations of SO quantifiers of the form $\exists R_1 \forall R_2 \exists R_3 \forall R_4 \phi$ and formulas with 2 or less alternations starting with either $\forall$ or $\exists$.

We note that from the definitions it is not clear that for arbitrary K $NFSOL[K] = SOL[K]$ and even for K's where it is so the "transformation rules" which produce an $NFSOL[K]$ formula given an arbitrary $SOL[K]$ formula, might be K–specific.

Lemma 13. *(Syntactic Lemma) Every formula in $INSOL[K]$ has an equivalent formula in $NFSOL[K]$.*

Proof: Define all formulas of $NFSOL[K]$ as *elementary*. Clearly every formula of $INSOL[K]$ can be generated by applying some inductive rules of SOL to a set of elementary formulas. We show that after each application of such an inductive formation rule the resulting $INSOL[K]$ formula has an equivalent $NFSOL[K]$ formula. The lemma follows by induction.

The inductive formation rules of SOL are: $\neg$, $\wedge/\vee$, First order quantification and second order quantification.

- By definition we have that $\forall R\phi$ and $\exists R\phi$ are in $NFSOL[K]$.
- For $\neg\phi$ the equivalent formula is obtained by "pulling" the negation inside while changing each $\exists(\forall)$ quantifier on the way to $\forall(\exists)$.
- For $\phi \wedge \psi$ ($\phi \vee \psi$) we rename relation variables to ensure that no quantified relation variable appears in both ϕ and ψ, and then "pull" out all quantifications of second order variables.
- For first order quantization we observe that an $\exists x \exists R\phi$ is semantically equivalent to $\exists R \exists x\phi$. For $\exists x \forall R\phi$ where R is of arity n we observe that the formula $\exists \bar{R} \forall x\psi$ is semantically equivalent if $\bar{R}$ is a relation of arity $n+1$ and ψ is obtained from ϕ by replacing any appearance of $R(\mathbf{y})$ with $\bar{R}(x, \mathbf{y})$. We also observe that $\forall x \forall R\phi = \forall R \forall x\phi$ and that $\forall x \exists R\phi = \forall \bar{R} \exists x\psi$ where ψ is obtained as above. Hence a first order quantifier can always be "pulled" before a second order quantifier at the cost of increasing the arity.

By a finite number of applications of the above rules we obtain the desired $NFSOL[K]$ formula. □

To prove equivalence between a (sub) logic and a complexity class we must prove two direction:

- Model Checking - that every set of structures definable by a formula in the logic is recognizable by a machine in the complexity class.
- Expressibility - that every set of structures recognizable by a machine in the class is definable by a formula in the logic.

5.1 Model Checking

Lemma 14. *Every formula in $FOL[K]$ has a model checker in $\mathbf{P}^K$ (even in $\mathbf{L}^K$).*

Proof: Every formula if $FOL[K]$ is also a formula of $FOL[ATC, K]$ ($FOL[DTC, K]$) and the proof follows from the model checking theorems of [MP94]. □

Theorem 15. *Every formula in $\exists SOL[K]$ has a model checker in $\mathbf{NP}^K$, for every X every formula of $XSOL[K]$ has a model checker in $(\Sigma_n^P)^K$ or $(\Pi_n^P)^K$ where Σ_n^P or Π_n^P is the level of the polynomial hierarchy capturing $XSOL$, every formula in $NFSOL[K]$ has a model checker in $\mathbf{PH}$.*

Proof: For $\psi = \exists R\phi$ a formula in $existsSOL[K]$, let M be a machine in $\mathbf{NP}^K$ which operates as follows: First it nondeterministically generates the relation R and then using it as a substitution simulates the machine in $\mathbf{P}^K$ which model checks ϕ (where ϕ is regarded as a formula on an extended vocabulary $\tau \cup \{R\}$). As R is polynomial in the input structure and so is the model checking the operation takes polynomial time.

For arbitrary X we work by induction on the quantifiers: For $\psi = \forall R\phi$ we note that this formula is equivalent to $\neg\exists R\neg\phi$. As model checking $\exists R\neg\phi$ is in $\mathbf{NP}^K$, model checking of ψ is in $\mathbf{CoNP}^K$.

Induction hypothesis: For every sequence of alternating quantifiers $\exists X$ ($\forall X$) of length n model checking of $\exists XSOL[K]$ ($\forall XSOL[K]$) can be done is $(\Sigma_n^P)^K$ ($(\Pi_n^P)^K$).

Inductive step: From the induction hypothesis we get that for every sequence of alternating quantifiers $\exists \forall X$ ($\forall \exists X$) of length $n+1$, model checking of $\forall XSOL[K]$ ($\exists XSOL[K]$) can be done in $(\Pi_n^P)^K$ $(\Sigma_n^P)^K$).

Let the model checker of $\psi \in \exists \forall XSOL[K]$ guess a substitution for the outer most quantified existential relations. using this as a substitution copy the extended structure to the oracle tape and ask the oracle in $(\Pi_n^P)^K$. As a nondeterministic step was done and an oracle for $(\Pi_n^P)^K$ was used, this is a machine in $(\Sigma_{n+1}^P)^K$.

For $\psi \in \forall \exists SOL[K]$ we observe that $\psi = \forall \exists X\phi$ can be written as $\neg\exists \forall \bar{X}\neg\phi$. As model checking of this formula is just the complement over all τ structures of the set of models $\exists \forall X\neg\phi$ which is in $(\Sigma_{n+1}^P)^K$ this problem is in $(\Pi_{n+1}^P)^K$.

For every ψ in $NFSOL[K]$ there is some X such that $\psi \in XSOL[K]$ hence it has a model checker in the appropriate level of $\mathbf{PH}^K$. □

5.2 Expressibility

The proofs below are given as outlines, the details are similar to the proofs for the unenhanced logics.

Theorem 16. *$\exists SOL[K]$ is as expressive as $\mathbf{NP}^K$.*

Proof: Let M be a machine in $\mathbf{NP}^K$, then there is a number d such that for any input of size n, M makes no more than n^d moves. Without loss of generality we can assume that the first $n^d/2$ moves of M non-deterministically generate a string of bits on a special tape which we shall call the *non-det* tape, while the second $n^d/2$ moves M performs deterministically, using the bits on the non-det tape to resolve any non-determinism in its tables.

Claim: The predicate $ORDER(R)$, saying that a given relation R is an order relation on (tuples of) the universe elements of a structure is first order definable.

Using an arbitrary order relation we can encode steps of the computation of M as d–tuples over the universe of the input structure.

Observe that as n^d is a bound on the amount of information on a tape of M we can use a d–tuple of elements over the input structure also to represent a tape position. Therefore a d–ary relation can describe the state of a tape of M if the alphabet is binary. If the alphabet is larger or if we wish to encode the head position too, this can be achieved via a $d+1$–ary relation. Hence a $2d+1$–ary tuple can represent the evolution of the tape contents during a computation of M. Using 5 such relations R_{state}, R_{input}, $R_{non-det}$, R_{work} and R_{oracle} we can represent the complete evolution of a computation of M. (Actually R_{input} and R_{state} can be of arity $d+1$).

Claim: The answer of the oracle for K at a given computation step, is definable by a $FOL[K]$ predicate over R_{oracle} where the Q_K quantifier is outer most.

Claim: The predicate $VALID_M$ over the 5 relations saying that these relations indeed represent the evolution of the computation of M is $FOL[K]$ definable. Note that this includes saying that all 5 relations are valid encodings of their appropriate objects: For each step j each tape position has a unique and valid value, each tape has only one head position, and only one state is active at a time. It also includes saying that for step zero R_{input} encodes the input and the work and oracle tape are empty, all heads are properly positioned and M is in its initial state. Finally, we have to say that the computation states indeed evolve according to the transition table of M. Here we use the fact that M acts deterministically as we cannot represent non-determinism while insisting on each tape cell having a unique value using fixed arity relations. Also here we use the Q_K quantifier (for those transitions which are oracle consultation).

Claim: The predicate $ACCEPTING(\mathbf{x})$ saying that a $d+1$–tuple of first order variable $\mathbf{x}$ represents an accepting state of the computation is first order definable.

The set of all structures accepted by a machine $M \in \mathbf{NP}^K$ can be represented as an $\exists SOL[K]$ formula:

$$\exists R_{order}, R_{input}, R_{state}, R_{non-det}, R_{work}, R_{oracle}, \exists \mathbf{x} ORDER(R_{order}) \wedge$$

$$\wedge VALID_M(R_{order}, R_{input}, R_{state}, R_{non-det}, R_{work}, R_{oracle}) \wedge ACCEPTING(\mathbf{x})$$

□

Theorem 17. *For every X $XSOL[K]$ is as expressive as $(\Sigma_n^P)^K$ or $((\Pi_n^P)^K$ where Σ_n^P or Π_n^P is the level of the polynomial hierarchy captured by $XSOL$.*

Proof: For $\forall SOL[X]$ the result follows as $(\Pi_1^P)^K = \mathbf{CoNP}^K$ is all sets of structures whose complements are in $\mathbf{NP}^K$ hence they can be written in $SOL[K]$ as $\neg\phi$ where $\phi \in \exists SOL[K]$ but by the syntactic lemma above this formula is equivalent to an $\forall SOL[K]$ formula.

Induction step: Assume the theorem to be true for all levels $\leq n$ of the relativized hierarchy.

We show this implies that the theorem holds for the $n+1$'st level.

For $(\Sigma^P_{n+1})^K$, one can view a machine $M \in (\Sigma^P_{n+1})^K$ as having access to an oracle in $(\Sigma^P_n)^K \cup (\Pi^P_n)^K$. Following similar arguments to the proof for $\mathbf{NP}^K$ we get that the set of structures accepted by M can be defined via a formula

$$\exists R_{order}, R_{input}, R_{state}, R_{non-det}, R_{work}, R_{oracle}, \exists \mathbf{x} ORDER(R_{order}) \wedge$$

$$\wedge VALID_M(R_{order}, R_{input}, R_{state}, R_{non-det}, R_{work}, R_{oracle}) \wedge ACCEPTING(\mathbf{x})$$

Where the predicate $VALID_M$ now contains an expression for oracles answers in $(\Sigma^P_n)^K \cup (\Pi^P_n)^K$. By the induction hypothesis in this expression all second order quantifiers are outside any Q_K quantifier, hence the formula we obtained is in $INSOL[K]$. By the syntactic lemma we can replace this formula with an equivalent $NFSOL[K]$ formula without increasing the number of alternations of second order quantifiers. □

Theorem 18. *$NFSOL[K]$ is as expressive as $\mathbf{PH}^K$*

Proof: Let M be some machine in $\mathbf{PH}^K$ by definition it belongs to some level of that hierarchy hence there is a $NFSOL[K]$ formula which defines the set of structures it accepts. □

5.3 Proof of Main Results

Putting the above theorems together we get:

Theorem 19. (Capturing the relativized polynomial hierarchy)

(i) *$\exists SOL[K]$ captures $\mathbf{NP}^K$.*
(ii) *For every X $XSOL[K]$ captures the appropriate level $(\Sigma^P_n)^K$ or $(\Pi^P_n)^K$ of $\mathbf{PH}^K$ where Σ^P_n or Π^P_n is the level captured by $XSOL$.*
(iii) *$NFSOL[K]$ captures $\mathbf{PH}^K$.*

Theorem 20. *$SOL[K] = NFSOL[K]$ and hence $SOL[K]$ captures $\mathbf{PH}^K$ and has a prenex normal form.*

Proof: $NFSOL[K] \subseteq SOL[K]$ by definition. As $NFSOL[K]$ captures $\mathbf{PH}^K$ it remains to show that every formula in $SOL[K]$ has a model checker in $\mathbf{PH}^K$. We prove by induction.

Basis: Let ϕ be a formula containing no Q_K quantifiers, then model checking of ϕ can be done in $\mathbf{PH}$ - using results by Meyer-Stockmeyer.

Inductive steps: 1. Assume Φ be a set of formulas feasible for K such that each sub formula of Φ has a model checker in $\mathbf{PH}^K$ then $Q_K\Phi$ has a model checker in $\mathbf{PH}^K$. Proof: Generating the structure $\mathcal{A}_\Phi$ can be done via a polynomial number of model checkings (see [MP93], [MP94]). As each model checking of a sub formula of Φ can be done in $\mathbf{PH}^K$ the whole operation is in $\mathbf{PH}^K$. A single query to the K oracle then completes the model checking.

2. Assume ϕ and ψ are formulas having model checkers in $\mathbf{PH}$ then every formula obtained from them via one application of an SOL formation rule also has a model checker in $\mathbf{PH}^K$. Proof: For negation use the fact that the $\mathbf{PH}$ is closed to complement. For $\vee$ ($\wedge$) apply the model checkers of the components and accept if either (both) accept. For first order quantification iterate over all elements of the universe of the input structure. This increases the complexity of the model checking by no more than a factor of $|A|$, hence the resulting model checker remains in $\mathbf{PH}^K$. For existential second order quantification - non-deterministically produce the required relation and perform the model checking for ϕ using this relation as a substitution. For universal SO quantification use closure to negation. □

6 After Thoughts

What have we achieved so far ? Mainly we have shown support for our thesis that extending a logic with a Lindström quantifier for a set K, is analogous to relativizing a complexity class to an oracle for K. Specifically we have shown this for the Log-space bounded classes **L**, **NL** and **AL** and for each Σ and Π level of the polynomial hierarchy. Moreover in those cases where we have such results, they are obtained uniformly independent of the semantics of the oracle used.

In this thesis there are still two large "gaps":

One is that we have not captured the important class $\mathbf{P}^K$. We conjecture that the natural transducer class for **P** is captured by an appropriate version of fixed point logic (LFP or IFP logic) and hence this logic enhanced by a first order Lindström quantifier for arbitrary K will capture $\mathbf{P}^K$.

The other is that we have not captured the class $\mathbf{PSpace}^K$ with poly-size transducers. Again we conjecture that the PFP logic enhanced by a first order Lindström quantifier for arbitrary K would do the trick. However we also do not yet rule out the possibility that $SOL[HEX, K]$ will be sufficient. This however would lead to some dramatic results:

- $SOL[HEX] = FOL[HEX] = \mathbf{PSpace}$.
- $SOL[HEX, K] = \mathbf{PSpace}^{bK}$ where the b indicates that the size of each query is subject to the poly-space bound.
- As we have oracles such that $\mathbf{PSpace}^{bK} \not\subseteq \mathbf{PH}^K$ an appropriate version of our theorem 9 will show that $HEX \notin \mathbf{PH}$ and hence $\mathbf{PSpace} \not\subseteq \mathbf{PH}$.

Given the difficulty of the $\mathbf{PSpace} =? \mathbf{PH}$ question we suspect that step two of this approach will fail.

Disregarding these "gaps", what can we conclude from our results?

For example, we may inquire into the relationships between the logic $FOL[HAM]$ and $\exists SOL$ and the logic FOL[HEX] and full SOL. It is known that $FOL[HAM] = \mathbf{L}^{\mathbf{NP}}$ while $\exists SOL = \mathbf{NP}$ hence we have

$$\exists SOL \subseteq FOL[HAM]$$

Also, it is known that $FOL[HEX] = \mathbf{PSpace}$ while $SOL = \mathbf{PH}$ hence we have

$$SOL \subseteq FOL[HEX]$$

We now ask:

Question. What happens to these containment relationships when both "sides" are enhanced with the same arbitrary quantifier Q_K?

On aesthetic grounds one might like to conclude that the containment relationship remains, however besides being WRONG it also implies an easy proof of $\mathbf{P} \neq \mathbf{NP}$:

Consider K to be the Baker, Gill and Solovey ([BGS75]) oracle such that $\mathbf{P}^K \neq \mathbf{NP}^K$. Clearly

$$FOL[ATC, K] = \mathbf{P}^K \neq \mathbf{NP}^K = \exists SOL[K]$$

If $\exists SOL[K] \subseteq FOL[HAM, K]$ then we can deduce $FOL[ATC, K] \subset FOL[HAM, K]$ and using theorem 9 show that the HAM quantifier is not expressible in $FOL[ATC]$, hence $\mathbf{P} \neq \mathbf{NP}$! Similarly the oracle separating $\mathbf{PH}^K$ from $\mathbf{PSpace}^K$ can be used to show that $SOL[K] \subseteq FOL[HEX, K]$ implies $\mathbf{PH} \neq \mathbf{PSpace}$!

The correct observation is that we can have logics $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1 \subseteq \mathcal{L}_2$ but $\mathcal{L}_1[K] \not\subseteq \mathcal{L}_2[K]$ if in the original inclusion the weaker logic had the richer syntax. Specifically $\exists SOL[K] \not\subseteq FOL[HAM, K]$ and $SOL[K] \not\subseteq FOL[HEX, K]$.

A proof of this for the FOL vs. SOL is a slight variation of the proof of theorem 7 as given in appendix C.

What does this mean in terms of complexity classes? Taking $FOL[HEX]$ and SOL as an example we see that we captured two classes where one has a possibly richer set of acceptors while

the other has a provably richer set of transducers. As if the set of transducers is somehow "tied" to the syntax while the set of acceptors can be enhanced via Lindström quantifiers (in truth adding a quantifier also increases the class of transducers but not by as much as we could expect).

One might claim that this "independence" of the acceptors and transducers is an undesirable consequence of using "noncomputational methods" to define "computational complexity classes", however we believe the problem is inherent in the way we think about complexity classes: In support of this belief we bring up the rich controversy as to how one should define the set of transducers for space bounded complexity classes, and also a recent result by Chor, Goldreich and Hastad [CGH] which shows that $\mathbf{IP}^K \neq \mathbf{PSpace}^K$ for a random oracle. This together with Shamir's result ([Sh92]) that $\mathbf{IP} = \mathbf{PSpace}$. Shows that even when defining complexity classes by "purely computational devices" it is possible to obtain "natural" classes which contain the same set of acceptors but different sets of transducers.

We conclude by observing the sad consequence of this line of reasoning: The conclusion one can draw from the separation or collapse of relativized complexity classes about the unrelativized classes, are only as meaningful as the relationships between the acceptors and transducers in the class. If for the same class of acceptors different sets of transducers can be "naturally" chosen - relativized results can tell us very little about the set of acceptors.

A When γ–reductions = functional reductions?

Definition21 (The relation computed by a device). Let M be some computing device and let x denote an input for M.

(i) The *string relation computed by* M, denoted $M(x)$, is obtained by ignoring the F output for those inputs which have an alternative output: $M(x) = F$ if M returns only F for this input and $s \in M(x)$ if M can return $s \neq F$. For consistent devices s, if exists, is unique hence $M(x)$ is a function.

(ii) Let $\mathtt{w}$ be a fixed invertible encoding scheme such that $\mathtt{w}$ maps structures to strings and $\mathtt{w}^{-1}$ maps strings to structures (see [MP93] for a discussion on encoding schemes). Without loss of generality we assume that every possible string over Σ^* corresponds under $\mathtt{w}$ to some structure. (We choose $\mathtt{w}$ such that those strings which don't fit are mapped to the empty structure). The *τ–σ relation computed by* M (denoted by $R(x)$ or by abuse of notation as $M(x)$) is obtained from the string relation computed by

$$\mathcal{A}_\sigma = \mathtt{w}^{-1} M(\mathtt{w}(\mathcal{B}_\tau))$$

For a transducer $R(x)$ is a relation between τ–structures and σ–structures and for an acceptor it is a relation between τ–structures and the set $\{0, 1, F\}$.

Definition22 (D-Reduction–Relations). Let τ and σ be two vocabularies.

(i) A *τ–σ reduction relation* is a relation R from τ–structures to σ–structures. (For consistent devices this is a function).

(ii) Let K_1 be a set of τ–structures and K_2 be a set of σ–structures. Let R be a reduction. We say that *R γ–reduces K_1 to K_2* if for every $\mathcal{B}$, $(\mathcal{A}, \mathcal{B}) \in R$ implies $\mathcal{A} \in K_1$ iff $\mathcal{B} \in K_2$.

(iii) Let T be a set of computing devices and $\mathbf{D}(T)$ be a relational complexity class. We say that K_1 is *$\mathbf{D}$–γ–reducible* to K_2, (denoted $K_1 \propto_{\gamma\mathbf{D}} K_2$) if there is a transducer $M \in T$ whose τ–σ relation reduces K_1 to K_2.

Note that our definition is equivalent to the usual notion of γ–reductions (see [Joh90] or [GJ79]) as the requirement there that each (string) input must have at least one non failing computation, is expressed here in the fact that the F output is also mapped to some σ–structure (which is either in K_2 or not).

We wish to investigate the classes $\mathbf{C}(T)$ of Turing machines for which the following proposition is true:

Proposition23. *Let K_1, K_2 be sets of structures. Then $K_1 \propto_{\gamma\mathbf{D}} K_2$ iff $K_1 \propto_{\mathbf{D}} K_2$.*

Definition24 (Self Reflecting Machines). Let M be a Turing machine (without loss of generality having one work tape). Then M' is called the *self reflecting version of* M, if M' simulates M with the addition that after each step in the operation of M, M' records on a (without loss of generality) separate work tape the current computation state of M (this includes machine states, work tape contents, head positions and any other info. necessary to simulate M). When the next step of M is performed, the new computation state overwrites the previous computation state.

Definition25 (Self Reflecting Class of Machines). T is *self reflecting* if for every $M \in T$, its self reflection version, M', is also in T.

Definition26 (Time-Multiplicative classes). A class T of bounded resources Turing machines is *time multiplicative* if for every function $f(n)$ such that $Time(M) \leq f(n)$ satisfies the time bounds of T it is also true that $Time(M) \leq f(n) \times f(n)$ satisfies the time bounds of T.

Question. Can we prove $\mathbf{D}$ regular and time bounded implies $\mathbf{D}$ multiplicative ?

Examples of multiplicative classes include **P**, **NP** the polynomial hierarchy, **EXPTIME** and more.

Lemma 27. *Let T be a set of space or time bounded turing machines such that T is time multiplicative and $\mathbf{L} \subseteq \mathbf{C}(t)$, then T is self reflecting.*

Proof: Observe that the space required for machine states is constant, space required to duplicate the work tape is equal the space of the work tape and space required to duplicate the position of the work tape head is logarithmic in the size of the tape.(This leads to the **L** limitation as recording the position of the input tape head requires LogSpace).

Also observe that the time required for machine states is constant, the time required for tape contents is the size of the work tape which is bounded by the time invested so far in computation which also bounds the time required for head positions. Hence if M is bounded by time $O(f(n))$ then M' is bounded by time $O((f(n))^2)$. For time multiplicative classes we are still left inside the class.

As we have not used any other resources (such as random bits, oracle consultations etc.) any bound on such resources does not effect this property, unless more information has to be encoded into the computation state - such as contents of oracle tapes or auxiliary push down automata. □

Note some classes which are not self reflective:

- LogSpace classes enhanced with a pushdown stack - as the stack contents cannot be recorded in LogSpace and adding an extra stack would throw us out of this complexity class.
- Linear time - as it is not multiplicative.

Lemma 28. *Let T be a set of self reflective, time multiplicative Turing machines. Let $M \in T$, Let S be the set of valid string inputs for M and $s \in S$ and let $S_1(s)$ denote the set of strings encoding feasible computation states of M when invoked with s as input. Then the set*

$$ValidPairs \equiv (s_1, s_2) \; s.t. \; s_1 \in S \; and \; s_2 \in S_1(s)$$

is recognizable in T. Let $NotFail(s)$ denote the subset of $S_1(s)$ whose members encode computation states which can lead to a non failing computation of M. Then

$$NonFailPairs \equiv (s_1, s_2) \; s.t. \; s_1 \in S \; and \; s_2 \in NotFail(s)$$

is also recognizable it T.

Proof: For the first part use M_m a modification of the self reflecting version of M such that M_m operates using only the first element of the pair as input and after each recording of the computation step, the recorded string is compared to the second element of the pair. If a match is found M_m accepts and stops.

For the second part use a modification of the above M_m which instead of stopping continues to simulate M until it stops. To this compose the machine which maps F to F and all other outputs to 1.

note that the added time complexity is multiplicative, and the added state complexity is that of recording the position of the input tape head. Hence the modification is in T. □

Theorem 29. *Let K_1, K_2 be sets of structures. Let $\mathbf{D}(T)$ be self-reflective, time multiplicative and closed under negation then $K_1 \propto_{\gamma \mathbf{D}} K_2$ implies $K_1 \propto_{\mathbf{D}} K_2$.*

Proof: Let $M_\gamma \in T$ be the transducer generating the γ reduction. Without loss of generality we can assume a full ordering of the transition table of M_γ. Let M be a machine which simulates the self reflective version of M_γ with the following changes: Whenever M_γ performs a non-deterministic step, M chooses among feasible transitions as follows: If all transitions lead to failing computations M fails. If there are transitions which lead to non-failing computations M continues with the smallest (according to the order) among these.

Note that if **D** is closed under negation than by lemma 28 $NonFailPairs_M$ has a full recognizer in $\mathbf{D}(t)$ and hence identifying the minimal non fail transition is in $\mathbf{D}(T)$ and Thus $M \in \mathbf{D}(T)$.

Clearly M computes a functional reduction of K_1 to K_2. □

B Correct proofs of NL, AL results from MP94

B.1 Prerequisites

Definition 30 The nf (normal form) convention. We say that an Oracle using Turing machine M obeys the *nf convention* if it accesses its oracle in the following form.

- Access to the oracle is via a stack of oracle tapes which is initially empty.
- M has a set of distinguished machine states denoted by S_{start} such that whenever M is in a state $s \in S_{start}$ a new oracle tape is pushed into the stack. This oracle tape is write only and one way and initially it contains only blank symbols which are not part of the alphabet of M.
- During its operation, M can write symbols only on the topmost oracle tape of the stack. (If the stack is empty no symbols can be written. This has the effect that prior to any writing on the oracle tape M must pass through a state from S_{start}).
- There is a set of distinguished machine states denoted by S_{answer} such that whenever M enters a state $s \in S_{answer}$ the following happens in a single time unit: 1. The topmost oracle tape is popped. 2. The machine moves into some state $pnext(s)$ if the contents of the popped oracle tape (those symbols which where not blank) are a string in the oracle set, and it moves into some state $nnext(s)$ if this string is not in the oracle state. (w.l.o.g. given s, $pnext(s)$ and $nnext(s)$ are unique). If M enters a state $s \in S_{answer}$ when the stack is empty M immediately stops and rejects its input.
- Time is counted for M in the usual way. Specifically, moving from a state of S_{answer} to its s_1 or s_2 (an oracle query) takes one unit of time, and entering a states of S_{start} (pushing a query tape) takes one unit of time. (Writing a symbol on the oracle tape takes no additional time as it is counted as part of a regular transition).
- Space is counted for M as follows: If $t_1, t_2 \ldots t_m$ are the string contents of the stack of oracle tapes, then the space used by this oracle stack is $\sum_{i=1}^{m} Max(1, log(|t_i|))$ (where $|t_i|$ denotes the length of string t_i).
- M may be subject to additional bounds regarding its access to the oracle beside those which follow from the way we count space and time.

Notation 31. *When an oracle computation model explicitly allows the use of a stack we denote by (i) and (∞) the cases when the oracle stack is bounded by i and when the oracle stack is fixed to a constant which is dependent on the machine but not on the input size. Without any additional notation, the model assumes that the depth of the stack is bounded only as a consequence of the restrictions on size and time by the nf convention.*

Definition 32 Configurations and Full Configurations. A *configuration* of a machine M is an encoding of its input, its work tapes, its head positions and its finite control state. A *full configuration* of a machine M includes additionally the contents of the oracle stack tape of M.

Notation 33. *We denote configurations of M by c, thus C is the set of all configurations. C_{answer} the set of all configurations with $s \in S_{answer}$, $c_{initial}$ is the initial configuration of M and $c_{accepting}$ is the accepting configuration. For a configuration $c \in C_{answer}$ $pnext(c)$ and $nnext(c)$ are the (unique) next configurations and $CNEXT(c)$ is the set of next configurations of an arbitrary configuration c.*

W.l.o.g. we assume a single accepting configuration c_{accept} and a single rejecting configuration c_{reject}: i.e. when it is clear that M should accept or reject it (in **P** *time and logarithmic space) erases all its tapes and move all its heads to their initial positions before announcing the acceptance or rejection.*

We denote a full configuration by a pair (x, t) where x is a configuration and t is the contents of the oracle tape. (when it is clear that the oracle tape is empty we shall abuse the notation and denote a full configuration by its non oracle tape part.

Definition 34. Let G be a graph where each node describes a full configuration and there is a directed edge between two nodes (x_1, t_1) and (x_2, t_2) if M can in one step move from (x_1, t_1) to (x_2, t_2).

- We define R_{TC} to be a relation over full configurations such that $x, t \in R_{TC}$ iff the full configuration x, t is reachable from $c_{initial}$.
- We define R_{ATC} to be a relation over full configurations such that $x, t \in R_{ATC}$ iff the full configuration x, t is ATC reachable from $c_{initial}$.

B.2 The Case of $\mathbf{L}^K$

The proofs for this case in [MP94] are correct and hence are not repeated. A slightly different proof can be obtained by adopting the proofs of the more complex **NL** case.

B.3 The Case of $\mathbf{NL}^K$

Definition 35 The q model. We say that M is oracle using via the q model if $R_{TC}(x, t)$ **is a function from** x **to** t, that is Given x if there is a t such that $R_{TC}(x, t)$ then this t is unique.

Definition 36 $\mathbf{NL}^{qK}$. We say that M is in $\mathbf{NL}^{qK}$ if M is nondeterministic, uses space logarithmic in input size, has access to an oracle set K, and $R_{TC}(x, t)$ **is a function from** x **to** t.

Theorem 37. *If $\phi \in FOL[TC, K]$ with quantifier nesting bounded by n, then ϕ has a model–checker in $\mathbf{NL}^{q(n)(K)}$.*

Proof:

Lemma 38. *Let $\Phi = \langle \phi, \psi_1, \ldots, \psi_m \rangle$ be τ formulas k–feasible for σ in $\mathcal{L}[K]$ each having a model checker in $\mathbf{NL}^{q(i)K}$. Then the formula $Q_K \Phi$ has a model checker in $\mathbf{NL}^{q(i+1)K}$.*

Proof: We generate the structure $\mathcal{A}_\Phi$ on the oracle tape and accept or reject according to the oracle answer. Since given a structure $\mathcal{A}$ each bit in the encoding structure $\mathcal{A}_\Phi$ is unique, as these bits are written, in each step the contents of the oracle stack is uniquely determined (provided the model checkers of Φ are all of the q model).

The structure is generated as follows: First generate A_Φ the universe of $\mathcal{A}_\Phi$. This is done by iterating over all k tuples of A, using each tuple as a substitution for the free variables in ϕ. A model checking of ϕ determines if this tuple is in A_Φ (in this case a counter counting the number of elements in A_Φ is increased by 1). At the end of this step the number of elements is written on the oracle tape.

Next the relations are produced. For a relation of arity r we iterate over all $r \times k$ –tuples. Each tuple is checked if it belongs to the universe. If a tuple belongs, then we invoke the model checker of ψ_i (using this tuple as a substitution), and write a 1 or a 0 on the oracle tape depending on if the model checker accepts or rejects.

Note that to ensure we are within the q model in both above steps we must seek evidence for our decision: If a model checker accepts we write a 1 but if the model checker rejects we have to invoke a "co-model checker" (one which rejects all non satisfying structures but might reject some satisfying structures) before we write a 0 or conclude that a tuple is not in the universe of $\mathcal{A}_\Phi$. This can be done as **NL** is closed to negation.

Also note that in these steps we increase the stack size by 1. (As things are already written on the oracle tape a model checking of ψ_i in depth i results in a total depth of $i + 1$). The space we use is that of the various model checkers, plus that of writing tuples of fixed arity and hence the whole operation is in $\mathbf{NL}^{mpK}$. □

Definition 39. Let $\chi \in FOL[TC, K]$ be a τ–formula which contains a sub-formula θ of the form $\theta \equiv Q_K \Phi$ such that θ is not nested inside a Q_K quantifier. Let $\tau' = \tau \cup \{R\}$ be the vocabulary formed by adding to τ a relation symbol R of arity i, where i is the number of free variables in θ. We define χ' - the *θ-reduced* equivalent of χ as the τ'–formula which is formed by replacing every appearance of the sub-formula θ in χ by the relation symbol R (over the free variables of θ).

Lemma 40. *Let χ, θ and χ' be as above. If both θ and χ' have model checkers of complexity $\mathbf{NL}^{q(i)(K)}$, then so does χ.*

Proof: Let M_χ, the model checker of χ, simulate $M_{\chi'}$ the model checker of χ' with the modification that whenever $M_{\chi'}$ consults the relation R on its input tape, M suspends its simulation of $M_{\chi'}$ and starts simulating M_θ the model checker of θ. Upon the termination of M_θ, M_χ resumes its simulation of $M_{\chi'}$ using the acceptance or rejection by M_θ as the required bit. Note that as R is not nested within any Q_K quantifier, when M_θ is invoked, the oracle stack is empty and it is empty again when $M_{\chi'}$ is resumed, hence M_χ uses a stack depth of no more than i and we are guaranteed to remain in the q model as all "new" full configurations have an empty oracle stack. □

The proof is now completed via an induction on depth of the Q_K quantifier. For depth 0 we just have $FOL[TC]$ formulas and the theorem holds. For the inductive step we apply the two lemmas above. (Lemma 38 for sub-formulas where the quantifier is syntactically outer most and lemma 40 when it is not). □

Corollary 41. $FOL[TC, K] \subseteq \mathbf{NL}^{q(\infty)K}$.

Theorem 42 Expressibility. *Let $M \in \mathbf{NL}^{q(\infty)K}$ be a Turing machine which recognizes a set of τ–structures A, using an oracle for K (a set of σ–structures). Then there is a formula in $FOL[TC, K]$ which defines A.*

Proof:
Basis: [Imm87] If M requires no oracle consultations than A can be defined by a formula of $FOL[TC]$.
Inductive Step: Assume every set of structures recognizable by a machine M of stack depth bounded by i can be defined by a formula of $FOL[TC, K]$ with $2i$ alternations of the TC and K quantifiers, then for every A recognized by an M which uses a stack of depth $i+1$, there is a defining formula which uses at most $2i+2$ alternations of the TC and K quantifiers.

Claim 43. *The predicates $E_0(x, y)$, $PNEXT(x, y)$, $NNEXT(x, y)$ saying that y is reachable from x in one step, without oracle consultation, with an accepting oracle consultation and with a rejecting oracle consultation, can be written as formulas of $FOL[TC]$.*

Definition 44. We define the languages L_0, L_1 and L_2 which are composed of tuples of the form: a τ structure, a configuration in C_{answer} and for L_1 and L_2 an integer representing a position on the oracle tape.

(i) $\mathcal{A}, y \in L_0$ iff on input $\mathcal{A}$ there is a configuration $x \in C_{start}$ of M such that starting with an empty oracle stack and in configuration x, the machine M (recognizing A) can eventually reach configuration y via a valid path of depth i.
(ii) $\mathcal{A}, y, j \in L_1$ iff $\mathcal{A}, y \in L_0$ and when y is reached the oracle tape contains exactly j bits.
(iii) $\mathcal{A}, y, j \in L_2$ iff there is a $k \geq j$ such that $\mathcal{A}, y, k \in L_1$ and when y is reached the j'th bit on the oracle tape is 1.

Claim 45. *L_0, L_1 and L_2 are all in $\mathbf{NL}^{q(i)K}$.*

In all three cases let the recognizing machine guess x and simulates M from configuration x keeping track of the validness of the path by counting excess of opening over closing parenthesis. During this simulation nothing is written for the bottom most query. For L_1 and L_2 we also keep track of the head position and for L_2 when the j'th position is written, its contents are remembered.

If during this simulation, the parenthesis count drops below 1 (the query started by x was answered) or rises above i before y was reached - reject. If y is not reached also reject. If y is reached stop the simulation. For L_0 accept, for L_1 accept if the "tape position counter" matches j and for L_2 accept if the "tape position counter" is greater or equal j and the remembered bit is a 1.

Corollary 46. *There are formulas $VALIDP_i(\mathcal{A}, y)$, $SIZE_i(\mathcal{A}, y, j)$ and $R1_i(\mathcal{A}, y, j)$ which define L_0, L_1 and L_2.*

Notice that the vocabulary for these formulas extends τ by appropriate tuples of constant symbols.

Claim 47. *The formulas*

$$\Phi_{string}(y) \equiv \langle \exists j (a \leq j) \wedge SIZE_i(y, j), R1_i(y, b) \rangle$$

where a, b are tuples of distinguished free variables, j is a tuple of bound variables, and y is a tuple of constant symbol, is feasible (over τ extended by a tuple of constant symbols) for the vocabulary binary strings *with $R1_i$ defining the unary relation of the bits which are '1'.*

Claim 48. *For y such that $VALIDP_i(\mathcal{A}, y)$, the formulas $Phi_{string}(y)$ define t the string on the oracle tape at configuration y.*

Observe that this claim is meaningful only as t is unique ($R_M(y, t)$ is a function).

Claim 49. *The formula*

$$ANEXT_i(y, c) \equiv VALIDP_i(y) \wedge ($$
$$(Q_K \Phi_{string}(y) \wedge PNEXT(y, c)) \vee (\neg Q_K \Phi_{string}(y) \wedge NNEXT(y, c)))$$

where y and c are tuples of free variables, says that there is a configuration $x \in C_{Start}$ such that if M is invoked in x it can eventually reach y via a valid path of depth i and then in a single oracle consulting step move to configuration c. (Notice that we added one alternation here).

To define the set A recognized by M of stack depth $i+1$ we wish to define the graph G of full configurations of M and apply to it the TC quantifier. Note however that it is sufficient to consider a sub graph of G which includes only those full configurations reachable from $c_{initial}$. For these, our q model guarantees that given a configuration x there is a single reachable full configuration. therefore we can encode the contents of the oracle tape by the configuration. For nodes which do not require oracle consultation this is sufficient. For nodes which do require oracle consultation we must be able to produce the oracle tape contents as a formula to which the Q_K quantifier can be applied to determine the out going edges. This is done as follows:

Writing a formula in $FOL[TC]$ encoding all valid configurations is done as in [Imm87]. We write E as

$$E(x, y) \equiv E_0(x, y) \bigvee_{k=0}^{i} ANEXT_i(x, y)$$

To define the set recognized by M we say that $c_{initial}$ is connected over this graph to $c_{accepting}$

$$ACCEPT \equiv TC_{x,y}^{c_{initial}, c_{accepting}} V(a), E(b, c), x, y$$

Note the use of the TC quantifier, thus giving $2i+2$ alterations. □

B.4 The $\mathbf{AL}^K$ Case

Definition 50 $\mathbf{AL}^{qK}$**.** We say that M is in $\mathbf{AL}^{qK}$ if M is alternating, uses space logarithmic in input size, has access to an oracle set K, and if $R_{TC}(y,t)$ and y is a query answering configuration then also $R_{ATC}(y,t)$.

The proof of the model checking theorem is as for the **NL** case (as the reproduction of the structure for the Q_K quantifier always obeys the additional restriction). The expressibility theorem also follows on similar lines, as via the restriction we get that if c_3 is reachable from c_1 via c_2 than c_2 is reachable from c_1 and c_3 is reachable from c_2. Hence L_0, L_1 and L_2 are all well defined and in $\mathbf{AL}^{q(i)K}$. (Without the restriction they are not well defined for the **AL** case).

C $\mathbf{P}^K$ cannot be captured by a FOL–based logic

Theorem 51. *There is an oracle using machine M which runs in polynomial time, such that if $L(K)$ is the language recognized by M where the semantics of K is a parameter ($L(K)$ is a function of K the oracle used), then there is no one formula $\phi \in FOL[ATC,K]$ where the semantics of K is a parameter, such that for all K ϕ defines $L(K)$.*

Intuitively the proof of this follows Buss' example which separates $\mathbf{NL}^K$ from $\mathbf{P}^K$, the details are given below.

Definition 52. For an arbitrary oracle K, let $S(K)$ be the following infinite binary string: s_1 - the first bit of $S(K)$ is 1 if "0" $\in K$ and 0 otherwise. s_i - the i'th bit - is 1 if $s_1 \ldots s_{i-1} \in K$ and 0 otherwise.

Definition 53. For an arbitrary oracle K, the language $L(K)$ is defined as follows: $x \in L(K)$ iff the $|x|$'th bit of the sequence $S(K)$ is 1. (Alternatively one can define $L(K)$ only over unary strings).

Lemma 54. *There is one oracle using machine M which runs in polynomial time, such that when M is connected to an oracle K it recognizes $L(K)$.*

Proof: M keeps one tape T reserved for the bits of $S(K)$. In the first step it writes on this tape 0. It then iterates the following procedure: copy the contents of T to the oracle tape and query the oracle: if it accepts add a 1 to T else add a 0. In each iteration the head over the input tape is moved one position to the left, when all the input is scanned accept if the leftmost bit of T is a 1. □

Definition 55. For any number n, we partition the set of all oracles into (M,n)*–equivalence classes* by defining any two oracles K_1, K_2 to be in the same class iff $M(K_1)$ and $M(K_2)$ give the same answer to all inputs of size less than n.

Lemma 56. *There are exactly 2^n (M,n)–equivalence classes.*

Proof: Clearly one cannot have more than 2^n such classes and for any binary string of length n one can find a K such that this string is the prefix of $S(K)$. □

Definition 57. Let ϕ be a formula of $FOL[ATC,K]$ with K a parameter (the semantics of K are as yet undefined, but its vocabulary is fixed). For any number n, we partition the set of all oracles (of this fixed vocabulary) into (ϕ,n)*–equivalence classes* by defining any two oracles K_1, K_2 to be in the same class iff ($\mathcal{A} \models \phi(K_1)$ iff $\mathcal{A} \models \phi(K_2)$) for all structures $\mathcal{A}$ of size less than n (over the appropriate vocabulary).

Lemma 58. *There is a fixed integer d such that there are no more than n^d (ϕ,n)–equivalence classes.*

Proof: We prove by induction on the formation rules of ϕ. If ϕ is an elementary formula then the number of equivalence classes is exactly 1 (as Q_K is not used).

Assume the theorem to be true for any ϕ with less than i formation rules, then it is also true for any ϕ with i formation rules (w.l.o.g. we assume only 5 formation rules):

(i) Assume ϕ of the form $\neg\psi$ with n^d bounding the number of (ψ, n)–equivalence classes. Then the number of (ϕ, n)–equivalence classes is also bounded by n^d.

(ii) Assume ϕ of the form $\psi \wedge \theta$ with n^{d_1} bounding the (ψ, n)–equivalence classes and n^{d_2} bounding the number of (θ, n)–equivalence classes. Then the number of (ϕ, n)–equivalence classes is bounded by $n^{d_1+d_2}$.

(iii) Assume ϕ of the form $\exists x\psi(x)$ with n^d bounding the number of (ψ, n)–equivalence classes (when x is interpreted as a constant). Then the number of (ϕ, n)–equivalence classes is bounded by n^{d+1}.

(iv) Assume ϕ of the form $ATC\Psi$ with n^d bounding (ψ, n)–equivalence classes for all $\psi \in \Psi$. Then the number of (ϕ, n)–equivalence classes is bounded by n^{4k+3d}, where k is the vectorization index.)$n^k \cdot n^d$ possibilities of different V and A and $n^{2k} \cdot n^d$ of different E - to actually get new possibilities the free variables have to be in the scope of a Q_K quantifier).

(v) Assume ϕ of the form $Q_K\Psi$ with n^d bounding the (ψ, n)–equivalence classes for all $\psi \in \Psi$. Then the number of (ϕ, n)–equivalence classes is bounded by $2 \times n^{d+k} < n^{d+k+1}$.

Since each ϕ has only finitely many formation rules there is some fixed d such that there are no more than n^d (ϕ, n)-equivalence classes. □

Proof of theorem: Let M be as above and assume there is a ϕ such that $\phi(K)$ defines $L(K)$. Let d be a constant such that there are no more than n^d (ϕ, n)–equivalence classes. Let n be a number such that $n^d < 2^n$.

There are two oracles K_1 and K_2 such that both are in the same (ϕ, n)–equivalence class but in different (M, n)–equivalence classes. For these oracles there is a number $m \leq n$ such that for all inputs of size m M accepts with one oracle and rejects with the other, but either both satisfy or dissatisfy ϕ. A contradiction to the assumption that $\phi(K)$ defines $L(K)$. □

D The ABC Theorem

Theorem 59. *Let A, B, C, K be sets of structures over possibly different vocabularies, such that K is definable in $FOL[B, C]$ but not in $FOL[A, C]$. Then B is not definable in $FOL[A]$.*

Proof: Let ϕ_K denote the $FOL[B, C]$ formula which defines the set K. Over all possible sets of structures definable in $FOL[B, C]$ but not in $FOL[A, C]$, let K be the set of structures such that ϕ_K has a minimum number of formation rules, (any $FOL[B, C]$ formula requiring less formation rules has a semantically equivalent formula in $FOL[A, C]$) where w.l.o.g. we assume $FOL[B, C]$ to have only the five formation rules $\neg$, $\vee$, $\exists x$, Q_B and Q_C.

Clearly ϕ_K cannot be a term and cannot be of the forms $\neg\psi$ and $\psi_1 \vee \psi_2$.

Claim 60. *ϕ_K cannot be of the form $\exists x\psi(x)$*

Proof: Let σ be the vocabulary of ϕ_K. Consider the formula $\psi(c)$ over the vocabulary σ' which extends σ with one constant symbol c. If the set of σ'–structures defined by $\psi(c)$ cannot be defined in $FOL[A, C]$ we contradict the minimality of ϕ_K and if it can be defined by some formula $\psi'(c) \in FOL[A, C]$ then $\exists x\psi'(x)$ defines K where $\psi'(x)$ is formed from $\psi(c)$ by substituting the variable x for the constant symbol c, thus contradicting the undefinability of K in $FOL[A, C]$. □

Claim 61. *If $\phi_K = Q\langle\phi_0, \phi_1, \ldots, \phi_m\rangle$ (where Q is either Q_B or Q_C) then for each of the ϕ_i subformulas there is an equivalent formula in $FOL[A, C]$ such that for every possible substitution of the free variables the two formulas describe the same set of structures.*

Proof: Assume the contrary, then by replacing the free variables of ϕ_i with constant symbols over an extended vocabulary, we define in $FOL[B,C]$ a set of structures not definable in $FOL[A,C]$. But ϕ_i uses less formation rules than ϕ_K - a contradiction. □

Corollary 62. *ϕ_K cannot be of the form $Q_C\Phi$.*

Claim 63. *ϕ_K is of the form $Q_B\langle\phi_0,\phi_1,\ldots,\phi_m\rangle$ where all the ϕ_i's are terms.*

Proof: That ϕ_K is of the form $\phi_K = Q_B\Phi$ follows by elimination of all other formation rules. Assume that some ϕ_i is not a term and consider the formula

$$Q_B\langle\phi_0,\phi_1,\ldots,\phi_{i-1},R_i,\phi_{i+1},\ldots,\phi_m\rangle \wedge \forall\mathbf{x_i}R_i(\mathbf{x_i}) \longleftrightarrow \phi_i(\mathbf{x_i})$$

over a vocabulary σ' extending σ with the relation symbol R_i of arity equal the number of distinguished free variables ϕ_i. This new formula defines a set of σ'–structures K'. If K' is definable by some $\psi \in FOL[A,C]$ then so is K: substitute the $FOL[A,C]$ equivalent of ϕ_i in place of R_i in the formula ψ. (By claim 61 such an equivalent to ϕ_i exists). Hence K' is undefinable in $FOL[A,C]$. But as

$$\forall\mathbf{x_i}R_i(\mathbf{x_i}) \longleftrightarrow \phi_i(\mathbf{x_i})$$

can be defined in $FOL[A,C]$, we conclude that the set of structures defined by

$$Q_B\langle\phi_0,\phi_1,\ldots,\phi_{i-1},R_i,\phi_{i+1},\ldots,\phi_m\rangle$$

cannot be defined in $FOL[A,C]$, thus contradicting the minimality of ϕ_K. □

Therefore, the set of structures defined by $Q_B\langle R_0,R_1,\ldots,R_m\rangle$ cannot be defined in $FOL[A,C]$ and hence cannot be defined in $FOL[A]$. □

Remark. Similar arguments prove the following variations of the above theory:

(i) When FOL is replaced by SOL.
(ii) When FOL is replaced by Σ_i or Π_i for arbitrary i.
(iii) When FOL is replaced by $\mathcal{L}$ where $\mathcal{L}$ is either FOL, SOL, Σ_i or Π_i enhanced by some quantifiers: If for some sets $K_1\ldots K_m$ and $L_1\ldots L_n$ we have $\mathcal{L}[K_1\ldots K_m] \neq \mathcal{L}[L_1\ldots L_n]$ then there must be some L_i not expressible in $\mathcal{L}[K_1\ldots K_m]$.
(iv) When $FOL[A,B]$ and $FOL[A,C]$ are restricted to formulas where the Q_C quantifier cannot be nested within itself.

References

[AV88] S.Abiteboul and V.Vianu. Fixpoint extensions of first order logic and Datalog-like languages. In Proc. 4'th IEEE symp. on Logiuc in Computer Science. pp 71-79, 1988.

[BF85] J. Barwise and S. Feferman, editors. *Model-Theoretic Logics*. Perspectives in Mathematical Logic. Springer Verlag, 1985.

[BGS75] T.Baker, J.Gill and R. Solovay. Relativization of the **P**?**NP** question. In SIAM J. comp. Vol. 4 pages 431-442, 1975.

[Bü60] J. R. Büchi. Weak Second Order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66-92. 1960.

[Bus88] J.F. Buss. Alternations and space–bounded computations. *Journal of Computer and System Sciences*, 36:351–378, 1988.

[CKS] A.K.Chandra, D.C. Kozen and L.J. Stockmeyer. Alternation. In Journal of the ACM vol. 28 No. 1. pp. 114-133, 1981.

[CGH] B. Chor, O. Goldreich and J. Hastad. The Random Oracle Hypothesis is False. Technion dept. of CS TR # 631 1990.

[Ebb85] H.D. Ebbinghaus. Extended logics: The general framework. In *Model-Theoretic Logics*, Perspectives in Mathematical Logic, chapter 2. Springer Verlag, 1985.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial time recognizable sets. In R. Karp, editor, *Complexity of Computation*, volume 7 of *American Mathematical Society Proc*, pages 27–41. Society for Industrial and Applied Mathematics, 1974.

[GJ79] M.G. Garey and D.S. Johnson. *Computers and Intractability.* Mathematical Series. W.H. Freeman and Company, 1979.

[Imm87] aN. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, Aug 1987.

[Joh90] D.S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1, chapter 2. Elsevier Science Publishers, 1990.

[Ko89] K.I.Ko. Relativized Polynomial Time Hierarchies Having exactly k levels. SIAM j. Comp. Vol 18 pages 392-408, 1989.

[LL76] R.E. Ladner and N. Lynch. Relativization of questions about log–space reducibility. *Mathematical Systems Theory*, 10:19–32, 1976.

[MS72] A.R.Meyer and L.J.Stockmeyer. The Equivalence problem for regular expressions with squaring requires exponential time. Proc. 13'th Ann. Symp. on Switching and Automata Theory, IEEE, Long Beach CA, pages 125-129.

[MP93] J.A. Makowsky and Y.B. Pnueli. Computable quantifiers and logics over finite structures. To appear in 'Quantifiers: Generalizations, extensions and variants of elementary logic', M. Krynicki, M. Mostowski and L.W. Szczerba eds., Kluwer Academic Publishers, 1993.

[MP94] J.A. Makowsky and Y.B. Pnueli. Oracles and Quantifiers LNCS 832 pp. 189-222, Springer 1994.

[Sh92] A. Shamir. IP = PSpace. *Journal of the ACM*, 39,4:869-877, 1992.

[Ste93a] I.A. Stewart. Logical characterizations of bounded query classes I: Logspace oracle machines. *Fundamenta Informaticae*, 18:65–92, 1993.

[Ste93b] I.A. Stewart. Logical characterizations of bounded query classes II: Polynomial–time oracle machines. *Fundamenta Informaticae*, 18:93–105, 1993.

[Sto87] L. Stockmeyer. Classifying the computational complexity of problems. *Journal of Symbolic Logic*, 52(1):1–43, 1987.

[Wil86] C.B. Wilson. Parallel computation and the NC hierarchy relativized. In *Structure in Complexity Theory*, volume 223 of *Lecture Notes in Computer Science*, pages 362–382. Springer Verlag, 1986.

This article was processed using the LaTeX macro package with LLNCS style