Oracles and Quantifiers

Markus Frick
Institut für Mathematische Logik
Universität Freiburg
frick@sun2.mathematik.uni-freiburg.de

August 1997

Abstract

We describe a general way of building oracle Turing machines, thereby treating oracle queries as reductions. The model allows various types of reductions, which lead to the different oracle models known from the literature.
For model theoretic purposes we define a functional transducer model which, for many complexity classes, corresponds to logical interpretations. We thus get a notion of reduction (δ-reduction) which is equivalent to logical reductions. Finally, applying δ-reductions to our new oracle model, we obtain results such as: if a logic $\mathfrak{L}$ captures a complexity class $\mathbf{D}$, then $\mathfrak{L}(Q_K)$ captures $\mathbf{D}^K$.

1 Introduction

The subject of descriptive complexity theory is the characterization of complexity classes by means of logics. A logic $\mathfrak{L}$ *captures* a complexity class $\mathbf{C}$ if model checking of $\mathfrak{L}$-formulas is in $\mathbf{C}$, and if every property which is decidable in $\mathbf{C}$ can be expresssed by an $\mathfrak{L}$-formula.
After capturing common complexity classes by logics, further research tried to characterize relativized complexity classes and found Lindström quantifiers as an adequate logical counterpart to oracle query mechanisms. Grädel in [Grä89] proved the following result: FO(IFP, Q_K) captures $\mathbf{P}^K$, which shows a one to one correspondence between the adjunction of a quantifier Q_K to a given logic and the relativization of a complexity class to an oracle K.
But unfortunately this result cannot be carried over to space bounded classes (for example logspace classes) and the corresponding logics. The principal obstacle is the nesting of quantifiers. In $\mathbf{P}$ nested quantifiers can be evaluated by storing the contents of the oracle tape on the work tape. This is not feasible in logspace classes (where the oracle tape can be of polynomial size).
Due to this problem, Gottlob in [Got95] investigated the relatization of $\mathbf{L}$ to classes $\mathbf{C}$ of oracles for which the nesting of quantifiers did not create additional complexity. So he obtained capturing results like FO(DTC, Q_K) captures $\mathbf{L}^{\mathbf{C}}$ for a $\mathbf{C}$-complete problem K, preserving the usual oracle model. For this model he furthermore showed the impossibility of a general result like that of Grädel.

In contrast to this approach, Makowsky and Pnueli tried to get a general capturing result like the one of Grädel. So they had to leave the usual oracle model and thus they suggested a new model with a stack of oracle tapes to get rid of the nested quantifiers. Apart from the stack, they adopted the model of [RST84] which only allows deterministic oracle queries, i.e. the oracle tape must be written deterministically. Although they overcame the problem of nested quantifiers, they failed in the treatment of nondeterminism and alternation: Their model does not satisfy the capturing results stated in [MP94] for **NL** and **AL** (unless **L** = **NL** = **AL**).

In this work we solve this quite intricate case by suggesting a new oracle model[1]. In contrast to the usual models, oracle consultation procedures are strictly separated from the rest of the machine. This is accomplished by the usage of separate (eventually nondeterministic) transducers which generate the query strings. So the query itself corresponds to a reduction to the oracle language. Therefore we get different oracle models by the application of different notions of reductions to the general model.
Its relevance for descriptive complexity theory turns out by the usage of a machine theoretic counterpart of logical reductions, the so called δ-reduction. Using this reduction type, we obtain results like them claimed by Makowsky and Pnueli, but for a larger set of complexity classes, particulary for nondeterministic and alternating logspace.

2 Basic Definitions

2.1 Complexity Theory

In this section we use a standard notion of *Turing machines* as stated in [Pap94] or any other work about computational complexity. We divide Turing machines in two classes; the members of the first class are computational devices recognizing sets of words (*languages*), the others compute certain string relations. The former ones we call *acceptors*, the latter ones *transducers*. Besides these two kinds of machines we consider *oracle* (or *relativized*) Turing machines[2]. These are Turing machines (acceptors or transducers) enhanced by an additional oracle query mechanism.

For simplicity we assume in the sequel that every Turing machine stops after a finite number of steps (for most complexity classes this can be accomplished by using standard counting techniques without leaving the class). Furthermore we restrict the alphabet Σ to $\{0,1\}$. ϵ represents the empty word. For a word $x \in \Sigma^*$ the k'th letter is denoted by $x[k]$; $x\frac{a}{k}$ stands for the string x where the k'th letter has been replaced by a.
For an acceptor M we denote by $L(M)$ the set of words accepted by M. Let T be a transducer and $x \in \Sigma^*$. Then $T(x)$ denotes the set of all outputs of the accepting paths. If there is no accepting path we set $T(x) = FAIL$.
We conceive a *complexity class* as a set of Turing machines (acceptors, transducers and

[1] Actually it should be called a metamodel, since it is a model of models. But we will prefer the term *model*.

[2] These are often called *Turing transducers*. In our context this term would be confusing since there are relativized acceptors as well as relativized transducers.

relativized versions). In the remainder we restrict ourselves to classes which are determined by a given resource bound (space or time), particulary to **L**, **NL**, **AL**, **P**, **NP**, **PH** and **PSPACE**.
Note that if we only consider the acceptors contained in a complexity class, we can substitute them by their recognized languages. This will be done in the context of oracles and reductions.

Besides the well known many-one reduction we consider two further reduction types. Let T be a transducer and A, B be languages over Σ. Then

- $A \leq^{\gamma} B$ via T if $T(x) \subseteq B$ iff $x \in A$
- $A \leq^{\epsilon} B$ via T if $T(x) \cap B \neq \emptyset$ iff $x \in B$

The composition of a transducer T and an acceptor M will be denoted by $T \circ M$. Note that for nondeterministic transducers the language recognized by its composition with an acceptor depends on the used reduction type. For instance an ϵ-reduction yields: $T \circ M$ accepts x iff $T(x) \cap L(M) \neq \emptyset$.
Let **D** and **C** be complexity classes. $A \leq_{\mathbf{D}} B$ holds if there exists a transducer $T \in \mathbf{D}$ with $A = L(T \circ B)$. By $\mathbf{D}(\mathbf{C})$ we denote the closure of **C** under **D**-reductions (according to one of the reductions above).

An oracle Turing machine has an additional write-only tape, the *oracle tape*, which will be erased after a query. Various models have been suggested, which differ in the mode in which the oracle tape can be written and whether or not the oracle tape is subject to the resource restrictions.
The model from Ladner and Lynch [LL76] imposes no restricitions, whereas in [RST84] a model is suggested in which the tape must be written in deterministic manner. Both agree in that they ignore the size of the oracle tape. For a more detailed discussion the reader is referred to [Bus88] and [MP94]. In section 3 we will describe a new model which relativizes the most important relations between complexity classes (in contrast to [LL76]) while it still provides some kind of nondeterministic oracle access (in contrast to [RST84]).

2.2 Logics

By $\overset{r}{x}$ we denote an r-tuple $(x_1, \ldots, x_r)$; if the arity of the tuple is known or unimportant we often write $\bar{x}$. $\overset{r}{x}\overset{s}{y}$ denotes the $r{+}s$-tuple $(x_1, \ldots, x_r, y_1, \ldots, y_s)$. A *signature* $\tau = \{R_1, \ldots, R_k, c_1, \ldots, c_l\}$ is a set of relation symbols R_i, each with an associated arity r_i, and constants c_i. A structure $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_k^{\mathfrak{A}}, c_1^{\mathfrak{A}}, \ldots, c_l^{\mathfrak{A}})$ over τ, consists of a universe A, the relations $R_i^{\mathfrak{A}} \subseteq A^{r_i}$, and the constants $c_i^{\mathfrak{A}} \in A$. Since we only consider finite ordered structures we assume in general, that the universe is an initial segment of the natural numbers. Often we interpret tuples of elements of the universe as natural numbers according to the induced lexicographic order.
A (k-ary) *query* q is a map from structures (over some signature τ) to (k-ary) relations on these structures. The k-ary query defined by a formula $\varphi(\overset{k}{x})$ maps the structure $\mathfrak{A}$ to the relation $\varphi^{\mathfrak{A}}(\_) := \{\overset{k}{a} \mid \mathfrak{A} \models \varphi[\overset{k}{a}]\}$.

Let $\sigma = \{P_1, \ldots, P_m\}$ be a relational signature[3] where P_i is of arity p_i. An $(m+1)$-tuple $\Psi = [\psi_0(\overset{k}{x}_0), \psi_1(\overset{k \cdot p_1}{x_1}), \ldots, \psi_m(\overset{k \cdot p_m}{x_m})]$ of τ-formulas we call *interpretation* of width k from τ to σ. We write $\Psi \in \mathfrak{L}$ if all formulas $\psi_i \in \mathfrak{L}$. By $\mathfrak{A}_\Psi := (\psi_0^{\mathfrak{A}}(\_), \psi_1^{\mathfrak{A}}(\_) \cap (\psi_0^{\mathfrak{A}}(\_))^{p_1}, \ldots, \psi_m^{\mathfrak{A}}(\_) \cap (\psi_0^{\mathfrak{A}}(\_))^{p_m})$ it induces a map from τ-structures to σ-structures (we always require that $\psi_0^{\mathfrak{A}}(\_) \neq \emptyset$).
Let K be a class of σ-structures. For a logic $\mathfrak{L}$, we define the *Lindström extension* $\mathfrak{L}(Q_K)$ by closing the set of formulas of $\mathfrak{L}$ under the following formation rule: if $\Psi \in \mathfrak{L}(Q_K)$ is an interpretation as above, then $Q_K \bar{x}_0, \ldots, \bar{x}_m \Psi$ is a formula of $\mathfrak{L}(Q_K)$. The quantifier binds the occurrences of the variables $\bar{x}_i$ and all other free variables remain free. If the $\bar{x}_i$ are clear from the context then we will omit them. The maximal nesting of (Q_K-)quantifiers of a formula we call its (Q_K-)*rank*. The meaning is given by: $\mathfrak{A} \models Q_K \Psi$ iff $\mathfrak{A}_\Psi \in K$.
For two classes of structures A, B we define $A \leq_{\mathfrak{L}} B$ iff $A = \{\mathfrak{A} \mid \mathfrak{A} \models Q_B \Psi\}$, or equivalently $\mathfrak{A} \in A$ iff $\mathfrak{A}_\Psi \in B$, for an interpretation $\Psi \in \mathfrak{L}$. This reduction corresponds to a many one reduction in complexity theory.

For the definition of the different *transitive closure operators* we consider structure of the form $\mathfrak{A} = (A, G^{\mathfrak{A}}, V^{\mathfrak{A}})$ with binary G and unary V. The elements of $V^{\mathfrak{A}}$ are called *existential*, the others are called *universal*. The alternating transitive closure $ATC(\mathfrak{A})$ is the smallest superset of $G^{\mathfrak{A}}$ which satisfies for all $x, y \in A$: $(x, y) \in ATC(\mathfrak{A})$ if, and only if **(i)** x existential and there is an $z \in A$ with $(x, z) \in G^{\mathfrak{A}}$ and $(z, y) \in ATC(\mathfrak{A})$, or **(ii)** x universal and for all $z \in A$ with $(x, z) \in G^{\mathfrak{A}}$ we have $(z, y) \in ATC(\mathfrak{A})$, and there is at least one such z.
Let $\mathfrak{L}$ be a logic. The logic $\mathfrak{L}$(ATC) is obtained by closing $\mathfrak{L}$ under the following formation rule: if $\varphi(\overset{k}{x}, \overset{k}{y})$ and $\chi(\overset{k}{x})$ are of $\mathfrak{L}$(ATC) then $[\mathrm{ATC}_{\overset{k}{x}, \overset{k}{y}} \varphi(\overset{k}{x}, \overset{k}{y}), \chi(\overset{k}{x})]\ \overset{k}{s}, \overset{k}{t}$ is of $\mathfrak{L}$(ATC). This formula holds in $\mathfrak{A}$ iff $(\overset{k}{s}, \overset{k}{t}) \in ATC(A^k, \varphi^{\mathfrak{A}}(\_, \_), \chi^{\mathfrak{A}}(\_))$.
By syntactically restricting this formation rule to $[\mathrm{TC}_{\bar{x}, \bar{y}} \varphi(\bar{x}, \bar{y})]\bar{s}, \bar{t} := [\mathrm{ATC}_{\bar{x}, \bar{y}} \varphi(\bar{x}, \bar{y}), \bar{x} = \bar{x}]\bar{s}, \bar{t}$ and $[\mathrm{DTC}_{\bar{x}, \bar{y}} \varphi(\bar{x}, \bar{y})]\bar{s}, \bar{t} := [\mathrm{TC}_{\bar{x}, \bar{y}} \varphi(\bar{x}, \bar{y}) \wedge \forall \bar{z}(\varphi(\bar{x}, \bar{z}) \to \bar{y} = \bar{z})]\bar{s}, \bar{t}$ we get the logics $\mathfrak{L}$(TC) and $\mathfrak{L}$(DTC), respectively.

The connection between logics and complexity classes will be established by an encoding of structures (for example by coding relations as adjacency matrices). Note that encoding and decoding can be done by arithmetics (feasible in logspace). By this we can use ordered structures as inputs of Turing machines.
The complexity of a query q is the complexity of the associated membership relation $\bar{a} \in q(\mathfrak{A})$ for structures $\mathfrak{A}$ with respect to the size of its universe. We say that q is *computable* in $\mathbf{C}$ if there exists an acceptor of $\mathbf{C}$ which decides the membership relation of the query (for example an acceptor expecting inputs of the form $(\mathfrak{A}, \bar{a})$). On the other hand we call q *expressible* in a logic $\mathfrak{L}$ if there exists a formula $\varphi \in \mathfrak{L}$ with $q(\mathfrak{A}) = \varphi^{\mathfrak{A}}(\_)$ for all structures $\mathfrak{A}$. $\mathfrak{L}$ *captures* $\mathbf{C}$ (for short: $\mathfrak{L} \equiv \mathbf{C}$) if a query is expressible in $\mathfrak{L}$ iff it is computable in $\mathbf{C}$.

For capturing results (encoding and decoding) we need a result concerning arithmetical computations. In the following proofs we will use it tacitly.

Lemma 1 (Immerman). *For each $k \geq 1$ the arithmetic operations $+, \cdot$, and $\leq$ on n-tuples of the universe are definable in* FO(DTC).

[3] From now on the two signatures τ and σ are fixed.

3 The Machine Models

To avoid any ambiguity we define all models at machine level. All Turing machines consist of a finite control which is given by a finite set of states and a transition table. Furthermore they have a work tape and an input tape with a read-write head and a read-only head respectively. Transducers additionally dispose of a write-only output tape with a special writing mechanism, i.e. after writing a letter on the tape, the head automatically moves to the next cell.

3.1 Acceptors

We start with the common definition of an alternating acceptor as stated in [CKS81].

Definition 2. *An alternating* acceptor *M is a 5-tuple $(Z, \Sigma, \delta, z_0, \beta)$ where*

- *Z is a finite set of states*
- *Σ is a finite tape alphabet*
- *$\delta \subseteq (Z \times \Sigma \times \Sigma) \times (Z \times \Sigma \times \{L, R, N\}^2)$ is the transition relation*[4]
- *z_0 is the initial state*
- *$\beta : Z \to \{\vee, \wedge, \mathbf{accept}, \mathbf{reject}\}$ assigns to each state a specific behaviour*

*A state $z \in Z$ with $\beta(z) = \vee$ ($\wedge$,**accept**,**reject**) is called* existential (universal, accepting, rejecting*, respectively).*
A configuration *of M is an element of $Z \times \Sigma^* \times \Sigma^* \times \mathbb{N}^2$ representing the state of the finite control, the contents of the input tape and the work tape and the respective head positions. Configurations are called universal, existential, rejecting, accepting according to their state. The initial configuration is $g_{M,x,0} = (z_0, x, \epsilon, 0, 0)$.*

We call an alternating Turing machine *nondeterministic*, if there are only existential states. A nondeterministic machine with a functional transition relation we call *deterministic*.
The successor relation $\vdash$ on the set of configurations is defined as follows. Let $g = (z, x, u, p, q)$ be a configuration and $(z', a, \sigma, \tau) \in \delta(z, x[p], u[q])$. Then

$$g \vdash h = (z', x, v, p + \sigma, q + \tau) \text{ where } v = u\frac{a}{q}$$

We say that h is a *direct successor* of g and we denote the transitive closure of $\vdash$ by $\vdash^*$. Now we can define what it means that an alternating acceptor M accepts a string $x \in \Sigma^*$. For this purpose we define a labeling function γ on the set of configurations. Let g be a configuration:

$$\gamma(g) := \begin{cases} \bigwedge_{g \vdash h} \gamma(h) & , \text{ if } g \text{ is universal} \\ \bigvee_{g \vdash h} \gamma(h) & , \text{ if } g \text{ is existential} \\ 1 & , \text{ if } g \text{ is accepting} \\ 0 & , \text{ if } g \text{ is rejecting} \end{cases}$$

[4] The letters L,R,N stand for the directions left, right, neutral and will be used later in an artihmetic manner. For example in an arithmetic expression L stands for -1.

Note that γ is well defined, since we required that every path leads to an accepting or rejecting configuration. M *accepts* $x \in \Sigma^*$ iff $\gamma(g_{M,x,0}) = 1$. The definition of the resources (space and time) used during the run is straightforward.

The next theorem states standard capturing results concering logspace acceptors. We will give a sketch of the underlying ideas to prepare the reader for the next sections. For a more detailed proof the reader is referred to [Imm87] and [EF95].

Theorem 3 (Immerman). FO(DTC) $\equiv$ **L**; FO(TC) $\equiv$ **NL**; FO(ATC) $\equiv$ **AL**

Proof: Like for all capturing results, we split the proof into two parts. The first part (*expressibility*) consists of the description of a given machine by a logical sentence. In the second part (*computability*) we have to build a machine which decides, if a structure is a model of a given sentence.
Since it is quite obvious how to construct a model checker for transitive closure logics, we omit this part. We outline the expressibility part for nondeterministic machines. The deterministic and alternating cases are slight modifications of this proof.
Let M be a nondeterministic acceptor with space bound $k \log n$. Then, by definition, M accepts a structure $\mathfrak{A}$ if, and only if there exists a path $g_{M,\mathfrak{A},0} \vdash g_1 \vdash \cdots \vdash g_l$ of configurations, where g_l is an accepting one.
Describing a computation path of M we can omit the contents of the input tape which remains unchanged during the run. Configurations without the contents of the input tape we call *GIDs* (genuine instantaneous descriptions). Thus we reduced the original problem to the question, whether or not, there exists a path connecting the initial GID with an accepting GID.
The first step to describe that by logics, is to encode GIDs by tuples of the universe of $\mathfrak{A}$. The work tape contents can be viewed as the binary representation of a number between 0 and $n^k - 1$, where $n = |\mathfrak{A}|$. The same holds for the positions of the heads of the work tape and input tape, respectively. Thus, all that information can be encoded by an m-tuple for a sufficiently large natural number m.
Since there is a finite number of states it suffices to add one component to the tuple to encode a whole GID of M (for sufficiently large universes). Note that decoding these tuples can be done by simple arithmetics, and is therefore feasible in FO(DTC).
The transition relation δ consists of finitely many conditions. Therefore it can be described by an FO(DTC) formula $E_0(\bar{x}, \bar{y})$ which holds in a structure $\mathfrak{A}$ iff $\bar{y}$ is the encoding of a direct successor of the GID encoded by $\bar{x}$.
By the application of the transitive closure operator on the graph given by $E_0^{\mathfrak{A}}(\_,\_)$ we can determine if there is a path from the initial GID to an accepting one, which is equivalent to the question of the acceptance of M set on $\mathfrak{A}$. $\Box$

3.2 Transducers and Reductions

We now turn to the definition of a new functional transducer model. This model follows the intention that a transducer of a complexity class **D** (which is problably nondeterministic or alternating) should at least be able to decide queries in **D**. To accomplish this, we first restrict transducers to deterministic writing on the output tape and moves the respective head to the next cell. In addition we equip them with subroutines of $\mathbf{D}$ to obtain the full complexity of $\mathbf{D}$.

Definition 4. *A* δ-transducer T *is a 7-tuple* $(Z, \Sigma, \delta, z_0, \beta, M_{\odot}, snext)$ *where*

- Z, Σ *and* z_0 *are as in the case of an acceptor*
- $\delta : (Z \times \Sigma \times \Sigma) \to (Z \times \Sigma \times (\Sigma \cup \{\#\}) \times \{L, R, N\}^2)$ *is the transition* *function*
- $\beta : Z \to \{\vee, \mathbf{accept}, \mathbf{reject}, \mathbf{sub}\}$
- *for each* $z \in \beta^{-1}(\mathbf{sub})$ *there exists an acceptor* M_z
- $snext : \{yes, no\} \times \beta^{-1}(\mathbf{sub}) \to Z$

A state $z \in Z$ *with* $\beta(z) = \mathbf{sub}$ *is called a* subroutine state *(we write* $z \in \mathbf{sub}$*). A* configuration *of* T *is an element of* $Z \times \Sigma^* \times \Sigma^* \times \Sigma^* \times \mathbb{N}^3$, *where the new components represent the contents of the write-only output tape and its head position. The initial configuration is* $g_{M,x,0} = (z_0, x, \epsilon, \epsilon, 0, 0, 0)$.

By the requirement that δ is a function we assure that T produces a unique output. The successor relation $\vdash$ changes as follows: Let $g = (z, x, u, v, p, q, r)$ be a configuration and $(z', a, b, \sigma, \tau) \in \delta(z, x[p], u[q])$ with $z \notin \mathbf{sub}$ and $b \neq \#$. Then

$$g \vdash h = (z', x, u', v', p + \sigma, q + \tau, r + 1) \text{ where } u' = u\frac{a}{q}, v' = v\frac{b}{r}$$

That is, T works as usual but in addition writes b on the output tape. If $b = \#$ then the output tape and its head remain unchanged.
If $z \in \mathbf{sub}$ then

$$g \vdash h = \begin{cases} (snext(yes, z), x, u, v, p, q, r) & , \text{ if } M_z \text{ on } x|u \text{ accepts} \\ (snext(no, z), x, u, v, p, q, r) & , \text{ if } M_z \text{ on } x|u \text{ rejects} \end{cases}$$

where $x|u$ is the concatenation of x, $|$ and u. $| \notin \Sigma$ serves to distinguish the input x from the "parameter" u. Note that since T works deterministically (δ is functional) there is a unique output; so we denote by $T(x)$ the output (and not the set $\{T(x)\}$).

The subroutine acceptors M_z ($z \in \mathbf{sub}$) have their own work tape and the resources they use during the run are simply added to the resources used by the calling transducer. For instance, for logspace classes it follows, that a logspace bounded transducer with logspace subroutines is of logspace (although the subroutines are started with a probably longer input).
Besides this, note that the definition includes the possibility of subroutines equipped with an oracle mechanism. This feature will be used when we define transducers with access to an oracle. In general, we set $T \in \delta\mathbf{D}$ if **(i)** T satisfies the resource bounds which are given for $\mathbf{D}$ and **(ii)** all subroutines of T are in $\mathbf{D}$. This definition also covers the case of relativized classes.
Many-one-reductions performed by δ-transducers we call *δ-reductions.*

Remarks:
Since the **L** subroutines can be simulated in **L** there is no difference between the transducers of **L** and δ**L**.
In case of **NL** we certainly loose something due to the restriction to a unique output. This can be examined more closely in connexion with the different reduction types. For instance one can prove that each γ-reduction can also be performed by a δ-transducer. Obviously this is very unlikely for ϵ-reductions, since here we have $\mathbf{NL}(\mathbf{L}) = \mathbf{NP}$.
Finally, the δ-model provides a reasonable model for alternating transducers .

The relevance of the δ-model lies in the one to one relation between δ-transducers and logical interpretations for many complexity classes and the respective logics. But before we can prove this relationship, we need to discuss some difficulties concerning the execution of subroutines and their description by means of logical expressions.
Given a δ-transducer T with input x, let z be a subroutine state and M_z the associated acceptor. As defined above, this acceptor M_z will be set on $x|u$, where u is the contents of the work tape in the respective configuration. How can we translate the concatenation of strings to the case of machines with structures as inputs?
Since we restrict ourselves to logspace classes the worktape of T set on $\mathfrak{A}$ can be encoded by an element of A^k for some k. We could set M_z on the expansion $(\mathfrak{A}, \bar{a})$ where $\bar{a}$ represents the work tape contents. But in describing computation paths by formulas it turned out to be easier to expand $\mathfrak{A}$ by the whole GID of T (which clearly includes the work tape contents).
We come to the following agreement: w.l.o.g. M_z expects inputs of the form $(\mathfrak{A}, \bar{a})$ where $\bar{a}$ encodes a GID. Thus for an acceptor M_z we obtain a formula $sub_z(\bar{x})$ with: M_z accepts $(\mathfrak{A}, \bar{a})$ iff $\bar{a} \in sub_z^{\mathfrak{A}}(\_)$ for GIDs $\bar{a}$.

Theorem 5. *Let A and B be arbitrary classes of τ, σ-structures, respectively. If $\mathbf{C}$ is a logspace bounded class,and $\mathfrak{L}_{\mathbf{C}}$ a logic with $\mathfrak{L}_{\mathbf{C}} \equiv \mathbf{C}$, then*

$$A \leq_{\mathfrak{L}} B \text{ iff } A \leq_{\mathbf{C}}^{\delta} B$$

Proof: We need to show that for each logical interpretation $\Psi \in \mathfrak{L}_{\mathbf{C}}$ we find a δ-transducer $T \in \delta\mathbf{C}$ with

$$T(\mathfrak{A}) = \mathfrak{A}_{\Psi} \text{ for all } \tau\text{-structures } \mathfrak{A} \qquad (*)$$

and vice versa. Then by the definition of δ-reductions and logical reductions we obtain the Theorem. Like for acceptors we divide the proof of (*) into the two parts of expressibility and computability.

Expressibility:
Let T be a δ-transducer with $\mathbf{C}$ subroutines and an n^r time bound (caused by the logspace bound). Out goal is to define a formula $\varphi_{out}(\overset{r}{x})$ which in $\mathfrak{A}$ evaluates to true iff the $\overset{r}{x}$-th letter written on the output tape is 1 (under the assumption that T set on $\mathfrak{A}$ writes at least $\overset{r}{x}$ many letters). Using this formula and decoding the output by arithmetics (feasible in FO(DTC)) we can define the interpretation Ψ (compare [EF95], pp. 144,145) which satisfies (*).
In contrast to the case of an usual acceptor we have to take into account the execution of subroutines and their results. To describe the finite control of T in states $z \notin \mathbf{sub}$ we use a

formula $E_0(\bar{g}, \bar{h}, c) \in \mathrm{FO(DTC)}$ with: $\mathfrak{A} \models E_0(\bar{g}, \bar{h}, c)$ iff $\bar{h}$ is a direct successor of $\bar{g}$ and if $c = 1$ ($c = 0$) then T writes a letter (no letter) on the output tape during this step.
For all subroutine states z let $sub_z(\bar{z})$ be the formula expressing the same query as M_z. That means $\mathfrak{A} \models sub_z[\bar{a}]$ iff M_z accepts $(\mathfrak{A}, \bar{a})$ where $\bar{a}$ is the GID of the calling configuration. Note that we have $sub_z(\bar{z}) \in \mathfrak{L}_{\mathbf{C}}$. $state_z(\bar{g})$ is true if $\bar{g}$ encodes a GID in state z. $spnext_z(\bar{x}, \bar{y})$ ($snnext_z(\bar{x}, \bar{y})$) determines the next GID if M_z accepts (rejects). Using this formulas

$$\varphi_{next}(\bar{g}, \vec{x}, \bar{h}, \vec{y}) := \bigvee_{z \notin \mathbf{sub}} \Big[state_z(\bar{g}) \wedge \exists c (E_0(\bar{g}, \bar{h}, c) \wedge \vec{y} = \vec{x} + c) \Big]$$
$$\bigvee_{z \in \mathbf{sub}} \Big[state_z(\bar{g}) \wedge \vec{x} = \vec{y} \wedge \big(sub_z(\bar{g}) \wedge spnext_z(\bar{g}, \bar{h}) \vee \neg sub_z(\bar{g}) \wedge snnext_z(\bar{g}, \bar{h}) \big) \Big]$$

holds if T moves in a single step from $\bar{g}$ to $\bar{h}$ and if in $\bar{g}$ the output tape contains $\vec{x}$ letters, then in $\bar{h}$ T has $\vec{y}$ letters on its output tape.
Let $\varphi^*_{next}(\bar{g}, \vec{x}, \bar{h}, \vec{y}) := [\mathrm{DTC}_{\bar{t}\vec{u}, \bar{v}\vec{w}} \varphi_{next}(\bar{t}, \vec{u}, \bar{v}, \vec{w})] \bar{g}\, \vec{x}, \bar{h}\, \vec{y}$ be the deterministic transitive closure of φ_{next} then

$$\varphi_{out}(\vec{x}) := \exists \bar{g} \bar{u}_1 \bar{u}_2 \Big(initial(\bar{g}) \wedge E_0(\bar{u}_1, \bar{u}_2, 1) \wedge \varphi^*_{next}(\bar{g}, \vec{0}, \bar{u}_2, \vec{x}) \wedge write1(\bar{u}_1) \Big)$$

is the sought formula, where $write1(\bar{g})$ holds if T in the step, departing from GID $\bar{g}$ writes a 1 to the output tape.

Computability:
Let $\Psi = (\psi_0(\bar{x}_0), \ldots, \psi_m(\bar{x}_m)) \in \mathfrak{L}_{\mathbf{C}}$ be an interpretation of width d from τ to σ. Let $i = 1 \ldots m$. By assumption ($\mathfrak{L}_{\mathbf{C}} \equiv \mathbf{C}$) there exist acceptors $M_i \in \mathbf{C}$ with: M_i accepts $(\mathfrak{A}, \bar{a}_i)$ iff $\mathfrak{A} \models \psi_i[\bar{a}_i]$.
To determine the relations $P_i^{\mathfrak{A}_\Psi}$ of the σ-structure $\mathfrak{A}_\Psi$, the transducer T cycles over all tuples $(\bar{a}_1, \ldots, \bar{a}_{p_i})$ for which $\mathfrak{A} \models \psi_0[\bar{a}_j]$ $j = 1, \ldots, p_i$ holds (then $\bar{a}_j$ is an element of the new universe). If also $\mathfrak{A} \models \psi_i[\bar{a}_1, \ldots, \bar{a}_{p_i}]$ holds, then $(\bar{a}_1, \ldots, \bar{a}_{p_i}) \in P_i^{\mathfrak{A}_\Psi}$. Obviously all this, including the encoding of the structure $\mathfrak{A}_\Psi$ can be done in logspace using subroutine acceptors $M_i \in \mathbf{C}$, thus proving (*). $\square$

3.3 Oracles and Lindström Logics

We now turn to the definition of the oracle model. The model bases on the tight relation between oracle consultations and reductions. A usual oracle Turing machine M has an additional oracle tape, on which it writes the query strings. In our new model this procedure is replaced by **(i)** the execution of a transducer which generates the query string and **(ii)** the answer of the oracle according to a particular reduction type. By that, restrictions imposed on the query mechanism in earlier models will be replaced by a suitable notion of reduction. We formalize this as follows:

Definition 6. *An* oracle acceptor M *is a 7-tuple* $(Z, \Sigma, \delta, z_0, \beta, T_\odot, onext)$ *where*

- Z, Σ *and* z_0 *are as in the case of an acceptor*

- $\delta \subseteq (Z \times \Sigma \times \Sigma) \times (Z \times \Sigma \times \Sigma \times \{L, R, N\}^2)$ *is the transition relation*
- $\beta : Z \to \{\vee, \wedge, \mathbf{accept}, \mathbf{reject}, \mathbf{query}\}$
- *for each* $z \in \beta^{-1}(\mathbf{query})$ *there is a transducer* T_z
- $onext : \{yes, no\} \times \beta^{-1}(\mathbf{query}) \to Z$

A state $z \in Z$ *with* $\beta(z) = \mathbf{query}$ *is called a* query state *(*$z \in \mathbf{query}$*) and the corresponding transducer* T_z *is called* query-transducer. *A* configuration *of* M *is an element of* $Z \times \Sigma^* \times \Sigma^* \times \mathbb{N}^2$. *The initial configuration is* $g_{M,x,0} = (z_0, x, \epsilon, 0, 0)$.

Note that since the oracle language does not appear in the definition, it is not part of the machine. It can be viewed as a formal parameter like in the next definition of the successor relation $\vdash$, which certainly makes use of the oracle. To define the successor relation, we have therefore to fix an oracle language $K \subseteq \Sigma^*$ and a reduction type (in the following definition this reduction type is hidden behind the $\circ$; *see:* composition of a transducer and an acceptor). For **query** configurations the direct successors are defined as follows. Let be $g = (z, x, u, p, q)$, with $z \in \mathbf{query}$. Then

$$g \vdash h = \begin{cases} (onext(yes, z), x, u, p, q) & , \text{ if } x|u \in T_z \circ K \\ (onext(no, z), x, u, p, q) & , \text{ if } x|u \notin T_z \circ K \end{cases}$$

As in the case above, the query-transducers have their own tapes. Time and space usage are determined as in the case of the transducer.

Concerning δ-reductions we define the following classes. $\mathbf{D}^{\delta(n+1)K}$ consists of all oracle acceptors M which **(i)** satisfy the resource bounds given for $\mathbf{D}$ and **(ii)** have query-transducers of $\delta\mathbf{D}^{\delta(n)K}$. We denote the union of the classes $\mathbf{D}^{\delta(n)K}$ by $\mathbf{D}^{\delta(\infty)K}$.
Other classes using for example ϵ-reductions can be defined in the same way.

Theorem 7. *Let be* $\mathbf{C} \in \{\mathbf{L}, \mathbf{NL}, \mathbf{AL}\}$ *and* $\mathfrak{L}$ = FO(DTC), FO(TC), FO(ATC) *be the respective capturing logic. Then for an arbitrary class of structures* K*:*

$$\mathfrak{L}(Q_K) \textit{ captures } \mathbf{C}^{\delta(\infty)K}$$

We prove an auxiliary result, which immediately yields the theorem.

Lemma 8. *Let* $\mathbf{C}$, $\mathfrak{L}$ *and* K *be given as above. Then* $\mathbf{C}^{\delta(n)K}$ *is captured by the set of formulas of* $\mathfrak{L}(Q_K)$ *where the nesting of* Q_K*-quantifiers is limited to* n.

Proof: The proof proceeds by induction on n. The *induction base* $n = 0$ is given by Theorem 3.
Induction step: $n \to n + 1$
As usual we have to show the two directions of expressibility and computability.
Expressibility:
Let $M \in \mathbf{C}^{\delta(n+1)K}$ and z be a query state. By Theorem 5 and the induction hypothesis there exists an interpretation $\Psi_z(\bar{x})$ which describes the same function as the query transducer T_z

(see (*) in the last proof). By taking a query as a single step we define the successor relation by

$$\varphi_{next}(\bar{g},\bar{h}) := \bigvee_{z\notin\mathbf{query}} (state_z(\bar{g}) \wedge E_0(\bar{g},\bar{h})) \bigvee_{z\in\mathbf{query}} \big[state_z(\bar{g}) \wedge \big((Q_K\Psi_z(\bar{g}) \wedge opnext_z(\bar{g},\bar{h})) \vee (\neg Q_K\Psi_z(\bar{g}) \wedge onnext_z(\bar{g},\bar{h}))\big)\big]$$

where $opnext, onnext$ define the next GID according to the answer of the oracle. By the application of the suitable TC-quantifier on the graph $\varphi^{\mathfrak{A}}_{next}(\_,\_)$ we get the formula expressing the same query as M. For instance, in case of $\mathbf{C} = \mathbf{NL}$ we get the sentence:

$$\varphi_M := \exists\bar{s}\bar{t}\,(initial(\bar{s}) \wedge accepting(\bar{t}) \wedge [\mathrm{TC}_{\bar{x},\bar{y}}\varphi_{next}(\bar{x},\bar{y})])\,\bar{s},\bar{t}$$

where $initial(\bar{s})$ ($accepting(\bar{s})$) is true if $\bar{s}$ encodes the initial (an accepting) configuration.

Computability:
Let $\varphi \in \mathfrak{L}(Q_K)$ with Q_K-quantifier rank $\leq n+1$. Let $\chi_i(\overset{k_i}{x}_i)$ be the outermost occurrences of formulas of the form $Q_K\Psi_i$ with free variables $\overset{k_i}{x}_i$. We replace them by new relation symbols L_i with arity k_i and obtain a formula $\varphi' \in \mathfrak{L}$ for which a model checker $M \in \mathbf{C}$ exists ($\mathbf{C} \equiv \mathfrak{L}$). By assumption the interpretations Ψ_i only contain formulas with Q_K-rank $\leq n$, and so by Theorem 5 there are transducers $T_i \in \delta\mathbf{C}^{\delta(n)K}$ defining the same function as Ψ_i. The new acceptor M' works as M, but instead of evaluating the relations L_i it performs a query using the transducer T_i. By the construction $M' \in \mathbf{C}^{\delta(n+1)K}$ decides the query described by φ. □

Note that the restriction to a certain logic is essential to the proof. So the statement of the Theorem does not hold for $\mathfrak{L} = \mathrm{FO(IFP)}$, although we have $\mathrm{FO(IFP)} \equiv \mathbf{AL}$. Particulary there exists an oracle K with $\mathrm{FO}(\mathrm{ATC}, Q_K) \neq \mathrm{FO}(\mathrm{IFP}, Q_K)$ (see [Imh96]). Thus, the assumption $\mathbf{C} \equiv \mathfrak{L}$ would not suffice for the Theorem.

4 Remarks

The results presented in the last section have first been stated by Makowsky and Pnueli. But as already mentioned, their machine model is too weak to provide model checkers for formulas containing subformulas of the form $Q_K\Psi$, where $\Psi \in \mathrm{FO(TC)}$. The reason for this lack is their restriction to deterministic queries. In terms of our new model we can describe this as follows: their model only allows queries which can be performed by simple many-one reductions. Unless $\mathbf{L} = \mathbf{NL}$, this is not sufficient to compute the FO(TC)-interpretations contained in the kernel of Lindström quantifiers. Thus their capturing claims for nondeterministic and alternating machines fail.
Our model overcomes this lack by use of δ-transducers. It furthermore provides a uniform way to extend results for acceptors to the case of transducers.

Finally, and as already indicated, we obtain usual oracle models by the application of suitable reduction types to the general model. For instance

(i) many-one reductions lead to the model suggested in [RST84] which only allows deterministic behaviour after the first letter was written on the oracle tape

(ii) For various deterministic and nondeterministic classes, the application of ϵ-reductions result in the model of [LL76]. This cannot be seen directly, since little tricks are needed to simulate a completely unrestricted behaviour through ϵ-reductions.

(iii) By using transducers with a restricted output tape we can also get the so called bounded oracle models.

References

[Bus88] J. Buss. Relativation. *Journal of Computer and System Sciences*, 36:351–378, 1988.

[CKS81] K. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 1995. ISBN 3-540-60149-X.

[Got95] G. Gottlob. Relativized logspace and generalized quantifiers over finite structures. *Proc. 10th IEEE Symp. on Logic in Computer Science*, pages 65–78, 1995.

[Grä89] E. Grädel. On logical descriptions of some concepts in structural complexity classes. In E. Börger, H. Kleine Buning, and M.M. Richter, editors, *CSL '89: 3rd Workshop on Computer Science Logic*, volume 440 of *LNCS*, pages 163–175. Springer-Verlag, 1989.

[Imh96] H. Imhof. *Fixed Point Logics and Generalized Quantifiers in Descriptive Complexity.* PhD thesis, Albert-Ludwigs-Universität Freiburg, 1996.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, August 1987.

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

[MP94] J.A. Makowsky and Y.B. Pnueli. Oracles and quantifiers. *Lecture Note in Computer Science*, 832:189–222, 1994.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, first edition, 1994.

[RST84] W. Ruzzo, J. Simon, and M. Tompa. Space bounded hierachies and probabilistic computations. *Journal of Computer and System Science*, 28:216–230, 1984.