On the Power of DNA-Computing*

Diana Rooß *Klaus W. Wagner*

Theoretische Informatik, Universität Würzburg

E-mail: {diana,wagner}@informatik.uni-wuerzburg.de

Abstract

In [Adl94] Adleman used biological manipulations with DNA strings to solve some instances of the Directed Hamiltonian Path Problem. Lipton [Lip94] showed how to extend this idea to solve any NP problem. We prove that exactly the problems in $\mathrm{P}^{\mathrm{NP}} = \Delta_2^{\mathrm{p}}$ can be solved in polynomial time using Lipton's model. Various modifications of Lipton's model, based on other DNA manipulations, are investigated systematically, and it is proved that their computational power in polynomial time can be characterized by one of the complexity classes P, Δ_2^{p}, or Δ_3^{p}.

1 Introduction

In the recent years several new ideas have been developed to use non electronic natural phenomena for real, efficient computation. In classical electronic-based computations the information is stored and modified bitwise by electric and electromagnetic means. It is typical for this kind of computations that the number of steps performed per time unit is huge but the number of processors running in parallel is small. The main objective for the new approaches mentioned above is not to speed up the number of steps per time unit but to increase the degree of parallelism considerably.

In 1985 David Deutsch [Deu85] proposed computers using quantum-physical effects to store and modify information. The quasi-probabilistic physical effect of "quantum parallelism" allows to construct quantum algorithms that solve certain problems faster than any known probabilistic algorithm. In [Deu85] *Quantum Turing Machines* are introduced as a theoretical model of such a kind of computation. In [Sho94] quantum machine algorithms for the discrete logarithm and for integer factoring are given which run in polynomial time. For both of these problems, no sequential polynomial time algorithms are known. In 1992 Leslie Valiant pointed out (see [BV93]) that the class of problems solvable in polynomial time by Quantum Turing Machines is included in P^{PP}.

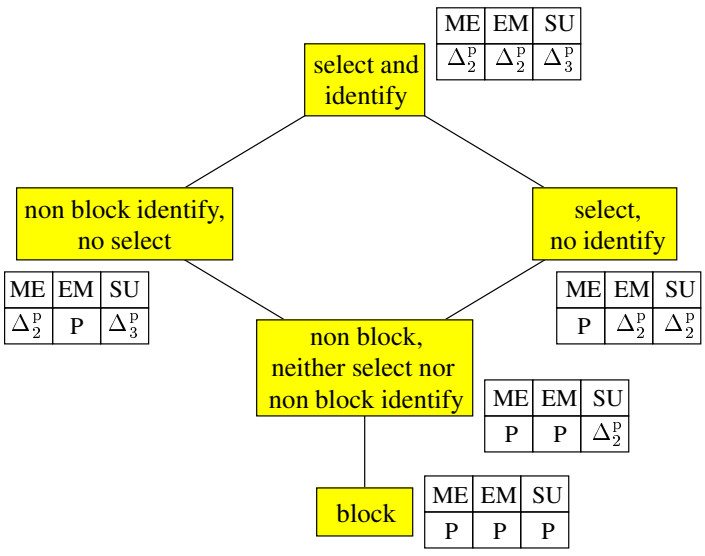
In 1994 different approaches came up that used biological properties of DNA strings to store and modify information. The general idea is to use a large number of DNA strings as

*Revised version of the technical report *On the Power of Bio-Computers* from March 1995. To appear in Information and Computation.

"processors" which compute in parallel. In [Pud94] Pavel Pudlák introduced *Genetic Turing Machines* that are probabilistic machines which can simulate the evolution of a population of strings using two special operators controlling the inheritance and the survival of strings. In this model on each of the randomly chosen paths *one* string is processed. It is proven in [Pud94] that the class of problems which can be solved by Genetic Turing Machines in polynomial time coincides with the class PSPACE, the class of problems solvable by sequential algorithms with polynomially bounded memory. Thus, the full power of parallelism is incorporated in the model of Genetic Turing Machines.

Also in 1994, Leonard Adleman [Adl94] used biological experiments with DNA strings to solve some particular instances of the *Directed Hamiltonian Path Problem* which is considered to be intractable because of its NP-completeness. In [Lip94] Richard Lipton showed how to extend this idea to solve any NP problem and discussed the practical relevance of this approach. He defined a model of biological computing that has, besides the classical means, the ability of manipulating large collections of DNA strings (in test tubes). Performing one of the special operations on a test tube means some simple manipulation of each of the strings in the test tube. In that way each DNA string corresponds to a piece of information, and all these pieces can be modified in parallel.

In the present paper we characterize the power of Lipton's model and some modifications of it. Our main issue is to develop Adleman's and Lipton's ideas as an interesting new theoretical approach to parallel computing: the manipulation of large sets of strings. Therefore we introduce in Section 2 the programming language DNA-Pascal as a model of DNA-computing. The various versions of DNA-Pascal differ by the collection of operations and tests on test tubes (that can simply be seen as set variables). In Section 3 we prove that the class of problems which can be solved in polynomial time via Lipton's model coincides with $\mathrm{P}^{\mathrm{NP}} = \Delta_2^{\mathrm{p}}$. (For basic notions in complexity theory see for example [BC94].) Some crucial properties characterize the computational power of set operations (see Section 4): namely the *block property*, the *select property*, and the *identify property*. For given combinations of those properties of operations, the following figure indicates the computational power of polynomial time DNA-Pascal when using the membership test (ME), the emptiness test (EM), or the subset test (SU).



In Section 5 and Section 6 one can find the proofs of the lower and upper bounds that lie

behind the overview of the computational power of the various selections of set operations and set tests.

2 DNA-Pascal: a model of DNA-computing

To describe algorithms for DNA-computers we introduce DNA-Pascal, a kind of reduced Pascal operating only on variables and arrays for non-negative integers supplied by a new type of variables for finite sets of words and some special instructions including these variables. By one of the usual one-to-one encodings we identify natural numbers with words over a finite alphabet Σ. Thus we refer to variables for natural numbers also as to word variables. In the context of genetics and following Adleman's and Lipton's ideas, $\{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ should be the alphabet of our choice. However, for simplicity we use the alphabet $\{0, 1\}$. Notice that all results below do not depend on the choice of the alphabet.

The different types of DNA-computers we will consider use in addition to the classical Pascal instructions the following instructions with set operations and conditions with set tests. Let $k \geq 0$ and $m \geq 1$ be variables for natural numbers, let a, $b \in \{0, 1\}$, let x be a word variable and let T, T_1 and T_2 be set variables. Let $I(x) \in \{0, 1\}^*$ be the contents of the word variable x, and let $I(T) \subseteq \{0, 1\}^*$ be the contents of the set variable T in a given moment. We define the cut operation $\setminus$ by $\setminus av =_{\mathrm{df}} v$ and $\setminus\varepsilon =_{\mathrm{df}} \varepsilon$. The following table shows the instructions with set operations:

Abbr.	Name	Instruction	New contents $I(T)$ of T
UN	Union	$T := T_1 \cup T_2$	$I(T_1) \cup I(T_2)$
IN	Initialization	$T := \mathrm{In}(k)$	$\{0, 1\}^k$
CO	Concatenation	$T := T_1 \cdot T_2$	$I(T_1) \cdot I(T_2)$
LC	Left Cut[1]	$T := \setminus T_1$	$\{ \setminus z \mid z \in I(T_1) \}$
SW	Subword Building	$T := \mathrm{Sw}(T_1)$	$\{ y \mid \exists v \exists w \, (vyw \in I(T_1)) \}$
LA	Left Adding[1]	$T := a \cdot T_1$	$\{a\} \cdot I(T_1)$
EQ	Bit Equalization	$T := \mathrm{Eq}(T_1, m, a)$	$\{vaw \mid (v0w \in I(T_1) \vee v1w \in I(T_1)) \wedge \lvert v\rvert = m-1\}$
BS	Bit Swap	$T := \mathrm{Bs}(T_1, m, a, b)$	$\{ vaw \mid v0w \in I(T_1) \wedge \lvert v\rvert = m-1 \} \cup \{ vbw \mid v1w \in I(T_1) \wedge \lvert v\rvert = m-1 \}$
BX	Bit Extraction	$T := \mathrm{Bx}(T_1, m, a)$	$I(T_1) \cap (\{0, 1\}^{m-1} a \{0, 1\}^*)$
BR	Bit Replacement	$T := \mathrm{Br}(T_1, m, a, x)$	$\{ v\, I(x)\, w \mid vaw \in I(T_1) \wedge \lvert v\rvert = m-1 \}$
BL	Length-preserving Bit Replacement	$T := \mathrm{Bl}(T_1, m, a, b)$	$\{ vbw \mid vaw \in I(T_1) \wedge \lvert v\rvert = m-1 \}$

Note that we are going to study the most important properties of operations (with respect to their computational power), but we want to keep the number of operations considered here as small as possible. Thus we leave out all the subword operations that could be defined analogously to the bit operations (e. g. subword extraction with the syntax $T := \mathrm{Sx}(T_1, x)$ and the semantics $I(T) = I(T_1) \cap (\{0, 1\}^* \, I(x) \, \{0, 1\}^*)$, or subword replacement with the syntax

[1] Analogously we could consider Right Cut and Right Adding instead of Left Cut and Left Adding or cross versions or all of them. This however, in fact finally leads to the same results.

$T := \mathrm{Sr}(T_1, x, y)$ and the semantics $I(T) = \{\, v\, I(y)\, w \mid v\, I(x)\, w \in I(T_1) \,\}$). It turns out that these operations have the same computational power as their bit operation counter parts (which can be proved with little coding tricks).

The following table shows the conditions with set tests:

Abbr.	Name	Condition	Condition is interpreted to be true if
ME	Membership Test	$x \in T$	$I(x) \in I(T)$
EM	Emptiness Test	$T = \emptyset$	$I(T) = \emptyset$
SU	Subset Test	$T_1 \subseteq T_2$	$I(T_1) \subseteq I(T_2)$

The input of a DNA-Pascal program is a string or a tuple of strings from $\{0,1\}^*$ which are given as the initial value of a word variable (or several word variables, resp.). The other variables used in the program are initially empty, i. e. word variables have value ε and set variables have value $\emptyset$. The computation can be terminated by the instructions "accept" or "reject" with the obvious semantics.

To the time complexity of such programs: To avoid the production of very long strings in a few steps we adapt the idea of the logarithmic cost criterion. The execution time of an operation $T := \ldots$ is estimated as the length of the longest word in T after the execution of this operation. Since we assume the set variables to be initially empty, it is evident that during a polynomially time-bounded operation every string in a set variable is polynomially length-bounded.

Notice that we do not need a special operator for assignment, because $T := T_1 \cup T_1$ has the same effect as $T := T_1$ (what we will use for short in the following). Nevertheless it is worth stating that copying a test tube is considered to be a realistic manipulation (see [Lip95]).

In this paper we are interested in DNA-Pascal programs that run in polynomial time and use different collections of set operations and tests. Therefore we define

Definition 1 Let $\mathcal{O}$ and $\mathcal{T}$ be collections of set operations and set tests, resp. DNA$(\mathcal{O}, \mathcal{T})$-P is the class of problems that can be decided in polynomial time via DNA-Pascal programs which use set operations from $\mathcal{O}$ and set tests from $\mathcal{T}$.

Obviously, these classes are closed under polynomial time Turing reducibility.

Propositon 2 $\mathrm{P}^{\mathrm{DNA}(\mathcal{O},\mathcal{T})\text{-P}} = \mathrm{DNA}(\mathcal{O}, \mathcal{T})\text{-P}$ *for every collection $\mathcal{O}$ of set operations and every collection $\mathcal{T}$ of set tests.*

3 The power of Lipton's model

In [Lip94] Richard Lipton made a first step to formalize Adleman's ideas of computing with DNA strings [Adl94]. He considered the following operations and tests to be realistic according to the possibilities to manipulate DNA strings in test tubes (which are set variables in DNA-Pascal):

- **union operation** (corresponds to our operation UN) which is not stated explicitly as an operation in [Lip94] but used implicitly,

- **operation of initialization** (corresponds to our operation IN) which is not stated explicitly either as an operation in [Lip94] but used in this way,
- **extract operation** (corresponds to our operation BX),
- **amplify operation**, i. e. copying test tubes (this is reflected by our semantic of DNA-Pascal which allows to use set variables various times without destroying their values),
- **detect operation** (corresponds to our test EM).

This translation of Adleman's and Lipton's ideas of "computing with molecular biology" into the language of DNA-Pascal is a further step of formalization and will allow us to give a precise characterization of the power of this kind of computing. The class of problems which can be solved in polynomial time "via molecular biology" appears to be the class DNA({UN,IN,BX},{EM})-P.

In [Lip94] Lipton showed that the NP-complete problem 3-SAT is decidable by these means in polynomial time. We are now going to present Lipton's proof using DNA-Pascal.

Theorem 3 [Lip94] 3-SAT $\in$ DNA({UN,IN,BX},{EM})-P.

PROOF. Let $H(x_1, \ldots, x_n) \equiv \bigwedge_{i=0}^{k}(x_{\sigma(i,1)}^{a_{i,1}} \vee x_{\sigma(i,2)}^{a_{i,2}} \vee x_{\sigma(i,3)}^{a_{i,3}})$ be a propositional formula in CNF having three literals in each clause, $\sigma(i,j) \in \{1, \ldots, n\}$ and $a_{i,j} \in \{0,1\}$ with $x^0 = \neg x$ and $x^1 = x$. Note that $(x^a = 1) \leftrightarrow (x = a)$. The following program in DNA-Pascal decides in polynomial time whether $H \in$ 3-SAT or not:

```
begin
  T := In(n);                          (put all possible "assignment strings" in T)
  for i := 0 to k do
    begin
      T1 := Bx(T, σ(i,1), a_{i,1});    (select all strings from T satisfying x_{σ(i,1)}^{a_{i,1}})
      T2 := Bx(T, σ(i,2), a_{i,2});    (select all strings from T satisfying x_{σ(i,2)}^{a_{i,2}})
      T3 := Bx(T, σ(i,3), a_{i,3});    (select all strings from T satisfying x_{σ(i,3)}^{a_{i,3}})
      T := T1 ∪ T2; T := T ∪ T3        (collect all strings satisfying the clauses 0, 1, ..., i)
    end;
  if T = ∅ then reject else accept     (only assignments satisfying all clauses are left in T)
end
```

□

Now we are ready to show that the class of all problems that can be decided by Lipton's model, coincides with a well-known complexity class:

Theorem 4 DNA({UN,IN,BX},{EM})-P $= \Delta_2^{\mathrm{p}}$.

PROOF. $\supseteq$: This follows from a chain of inclusions that use Proposition 2 and Theorem 3:

$$\Delta_2^{\mathrm{p}} = \mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\text{3-SAT}} \overset{\mathrm{T3}}{\subseteq} \mathrm{P}^{\text{DNA(\{UN,IN,BX\},\{EM\})-P}} \overset{\mathrm{P2}}{=} \text{DNA(\{UN,IN,BX\},\{EM\})-P}.$$

$\subseteq$: Let L be a language decidable by a ({UN,IN,BX},{EM})-DNA-Pascal program P in time p where p is a polynomial. Let $T_1, \ldots, T_k$ be the set variables in that program. These set variables have an influence on the result of the computations only by emptiness tests. We show how to simulate the emptiness tests in the program. Therefore we define

$$R_P =_{\text{df}} \{(x, a_1 \ldots a_m, z, i, t) \mid \begin{array}{l} z \in T_i \text{ at step } t \text{ of program } P \text{ on input } x \text{ and} \\ \text{there have been } m \text{ tests before step } t \text{ and} \\ a_1, \ldots, a_m \in \{0,1\} \text{ are the results of these tests}\}. \end{array}$$

That R_P is in P can be seen as follows. To decide $(x, a_1 \ldots a_m, z, i, t) \in R_P$ we basically simulate the program P. We introduce new variables $s_1, \ldots, s_k$ for natural numbers with the intention that the value of s_j should be 1 if $z \in T_j$ in the actually simulated step and 0 otherwise. This is achieved by replacing the set instructions in P by Pascal instructions without set variables. Here is the table of replacements (let $z(m)$ be the m-th symbol in the word z):

set instruction	replaced by Pascal instruction
$T_j := T_u \cup T_v$	$s_j := \max(s_u, s_v)$
$T_j := \mathrm{In}(n)$	if $\|z\| = n$ then $s_j := 1$ else $s_j := 0$
$T_j := \mathrm{Bx}(T_u, m, a)$	if $z(m) = a$ then $s_j := s_u$ else $s_j := 0$

The r-th test ($r = 1, \ldots, m$) is replaced by the test "$a_r = 1$". Finally, after having simulated t steps of P in such a way, the answer to $(x, a_1 \ldots a_m, z, i, t) \in R_P$ is the value of s_i at this moment.

Now we are ready to describe the simulation of the ({UN,IN,BX},{EM})-DNA-Pascal program P by an ordinary Pascal program P' using a coNP-oracle. First observe that since the running time of P on x is bounded by $p(|x|)$ also the length of the words in the set variables are bounded by $p(|x|)$. Thus an emptiness test $T_i = \emptyset$ in step t of the computation of P on x can be replaced by the query

$$\neg\, \exists z \; (\; |z| \leq p(|x|) \wedge (x, a_1 \ldots a_m, z, i, t) \in R_P)$$

assumed that there have been exactly m tests before with the results $a_1, \ldots, a_m \in \{0,1\}$. This is clearly a question to a coNP-oracle. Since P' can collect the answers to all oracle queries already made (which are identically with the corresponding emptiness test results of P) the answer string $a_1 \ldots a_m$ needed for the next query is known.

Therefore we can decide L with a polynomially time-bounded Pascal program without set variables that asks queries to a coNP-oracle (or, equivalently, to an NP-oracle).

□

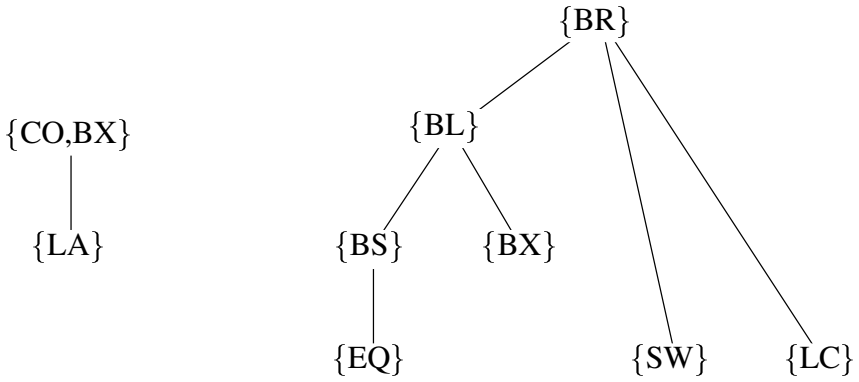
4 Different Sets of Operations and Tests

In this section we first discuss different groups of operations and tests and then we give a complete overview about the computational power of the various selections of operations and tests.

The operation of joining two test tubes is so natural and elementary that we will include the operation UN in every set of operations. In the same way it is assumed that the operation IN is always available, since without any initialization the initially empty set variables cannot really be used.

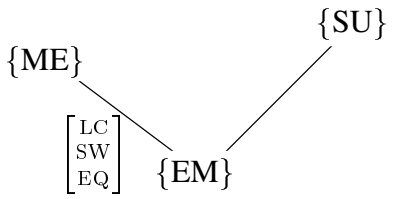
The following lemmas state some easy-to-prove relations between different operations and tests.

Lemma 5 *In the following graph an edge* $\begin{matrix}\mathcal{O}_2\\|\\\mathcal{O}_1\end{matrix}$ *in upward direction means that the operations from set* $\mathcal{O}_1$ *can be simulated by the operations from set* $\mathcal{O}_2$ *if* UN *and* IN *are available, i. e.* DNA($\mathcal{O} \cup \mathcal{O}_1$,$\mathcal{T}$)-P $\subseteq$ DNA($\mathcal{O} \cup \mathcal{O}_2 \cup \{\mathrm{UN}, \mathrm{IN}\}$,$\mathcal{T}$)-P *for all collections* $\mathcal{O}$ *of set operations and all collections* $\mathcal{T}$ *of set tests:*



Lemma 6 *For all collections* $\mathcal{O}$ *of set operations and all collections* $\mathcal{T}$ *of set tests,*

1. DNA($\mathcal{O}$,$\mathcal{T} \cup \{\mathrm{EM}\}$)-P $\subseteq$ DNA($\mathcal{O} \cup \{\mathrm{UN}, \mathrm{IN}\}$,$\mathcal{T} \cup \{\mathrm{SU}\}$)-P,
2. DNA($\mathcal{O}$,$\mathcal{T} \cup \{\mathrm{EM}\}$)-P $\subseteq$ DNA($\mathcal{O} \cup \{\mathrm{UN}, \mathrm{IN}, OP\}$,$\mathcal{T} \cup \{\mathrm{ME}\}$)-P *for* $OP \in \{\mathrm{LC}, \mathrm{SW}, \mathrm{EQ}\}$.



The following theorem gives a complete overview about the complexity classification of the classes DNA($\mathcal{O}$,$\mathcal{T}$)-P for all possible $\mathcal{O} \subseteq$ {UN, IN, CO, LC, SW, LA, EQ, BS, BX, BL, BR}, such that $\mathrm{UN}, \mathrm{IN} \in \mathcal{O}$, and all possible $\mathcal{T} \subseteq \{\mathrm{ME}, \mathrm{EM}, \mathrm{SU}\}$.

The term $\begin{bmatrix}A_1\\\vdots\\A_k\end{bmatrix}$ stands for an arbitrary nonempty subset of $\{A_1, \ldots, A_k\}$, wheras the term $\begin{bmatrix}-\\A_1\\\vdots\\A_k\end{bmatrix}$ stands for an arbitrary subset (including the empty set) of $\{A_1, \ldots, A_k\}$.

Theorem 7 *If the crossing of row $\mathcal{O}$ and column $\mathcal{T}$ in the following table contains the complexity class $\mathcal{K}$ then* DNA($\mathcal{O} \cup \{\mathrm{UN}, \mathrm{IN}\}$,$\mathcal{T}$)-P $= \mathcal{K}$.

$\mathcal{O}$ \ $\mathcal{T}$	$\left[\mathrm{ME}\right]$	$\left[\mathrm{EM}\right]$	$\left[\mathrm{ME}\right] \cup \left[\mathrm{EM}\right]$	$\begin{bmatrix}-\\ \mathrm{ME}\\ \mathrm{EM}\end{bmatrix} \cup \left[\mathrm{SU}\right]$
$\begin{bmatrix}-\\ \mathrm{CO}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LC}\\ \mathrm{SW}\end{bmatrix}$	$\mathrm{P}^{(U1)}$	$\mathrm{P}^{(U1)}$	$\mathrm{P}^{(U1)}$	$\mathrm{P}^{(U1)}$
$\begin{bmatrix}-\\ \mathrm{CO}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LC}\\ \mathrm{SW}\end{bmatrix} \cup \left[\mathrm{LA}\right]$	$\mathrm{P}^{(U2)}$	$\mathrm{P}^{(U2)}$	$\mathrm{P}^{(U2)}$	$\Delta_2^{\mathrm{p}}{}^{(U3)}_{(L1)}$
$\begin{bmatrix}-\\ \mathrm{CO}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LC}\\ \mathrm{SW}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LA}\end{bmatrix} \cup \begin{bmatrix}\mathrm{EQ}\\ \mathrm{BS}\end{bmatrix}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L2)}$	$\mathrm{P}^{(U4)}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L2)}$	$\Delta_3^{\mathrm{p}}{}^{(U8)}_{(L5)}$
$\begin{bmatrix}-\\ \mathrm{CO}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LA}\end{bmatrix} \cup \left[\mathrm{BX}\right]$	$\mathrm{P}^{(U5)}$	$\Delta_2^{\mathrm{p}}{}^{(U6)}_{(L3)}$	$\Delta_2^{\mathrm{p}}{}^{(U6)}_{(L3)}$	$\Delta_2^{\mathrm{p}}{}^{(U6)}_{(L3)}$
$\begin{bmatrix}-\\ \mathrm{CO}\end{bmatrix} \cup \begin{bmatrix}\mathrm{LC}\\ \mathrm{SW}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LA}\end{bmatrix} \cup \left[\mathrm{BX}\right]$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_3^{\mathrm{p}}{}^{(U8)}_{(L4)}$
$\begin{bmatrix}-\\ \mathrm{CO}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LC}\\ \mathrm{SW}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LA}\end{bmatrix} \cup \begin{bmatrix}\mathrm{EQ}\\ \mathrm{BS}\end{bmatrix} \cup \left[\mathrm{BX}\right]$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_3^{\mathrm{p}}{}^{(U8)}_{(L5)}$
$\begin{bmatrix}-\\ \mathrm{CO}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LC}\\ \mathrm{SW}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{LA}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{EQ}\\ \mathrm{BS}\end{bmatrix} \cup \begin{bmatrix}-\\ \mathrm{BX}\end{bmatrix} \cup \begin{bmatrix}\mathrm{BL}\\ \mathrm{BR}\end{bmatrix}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_2^{\mathrm{p}}{}^{(U7)}_{(L3)}$	$\Delta_3^{\mathrm{p}}{}^{(U8)}_{(L5)}$

The marks (Li) (and (Uj), resp.) indicate that the lower bound results $\mathcal{K} \subseteq$ DNA($\mathcal{O} \cup \{\mathrm{UN}, \mathrm{IN}\}$,$\mathcal{T}$)-P *(the upper bound results* DNA($\mathcal{O} \cup \{\mathrm{UN}, \mathrm{IN}\}$,$\mathcal{T}$)-P $\subseteq \mathcal{K}$*, resp.) follow, using Lemma 5 and Lemma 6, easily from Lemma Li (Lemma Uj, resp.) stated below and proved in Section 5 (Section 6, resp.).*

Note that 3-SAT is an NP-complete problem, and the problem 3-∀SAT which is defined as the set of all propositional formulas $H(x_1, \ldots, x_n, y_1, \ldots, y_m)$ in CNF with three literals per clause such that $\forall b_1 \ldots \forall b_m \exists a_1 \ldots \exists a_n \, (H(a_1, \ldots, a_n, b_1, \ldots, b_m) = 1)$ is Π_2^{p}-complete. Thus the membership of 3-SAT (3-∀SAT, resp.) in DNA(...)-P has the immediate consequence

$$\begin{array}{l} \Delta_2^{\mathrm{p}} = \mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\text{3-SAT}} \subseteq \mathrm{P}^{\text{DNA(...)-P}} = \text{DNA(...)-P} \\ (\Delta_3^{\mathrm{p}} = \mathrm{P}^{\Pi_2^{\mathrm{p}}} = \mathrm{P}^{\text{3-}\forall\text{SAT}} \subseteq \mathrm{P}^{\text{DNA(...)-P}} = \text{DNA(...)-P, resp.}) \end{array}$$

Lemma L1 3-SAT $\in$ DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{LA}\}$,$\{\mathrm{SU}\}$)-P.

Lemma L2 3-SAT $\in$ DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{EQ}\}$,$\{\mathrm{ME}\}$)-P.

Lemma L3 (Theorem 3) 3-SAT $\in$ DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{BX}\}$,$\{\mathrm{EM}\}$)-P.

Lemma L4 3-∀SAT $\in$ DNA($\{\mathrm{UN}, \mathrm{IN}, \begin{bmatrix}\mathrm{LC}\\ \mathrm{SW}\end{bmatrix}, \mathrm{BX}\}$,$\{\mathrm{SU}\}$)-P.

Lemma L5 3-$\forall$SAT $\in$ DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{EQ}\}$,$\{\mathrm{SU}\}$)-P.

Lemma U1 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{LC}, \mathrm{SW}\}$,$\{\mathrm{ME}, \mathrm{SU}\}$)-P $\subseteq$ P.
Lemma U2 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{LC}, \mathrm{SW}, \mathrm{LA}\}$,$\{\mathrm{ME}\}$)-P $\subseteq$ P.
Lemma U3 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{LC}, \mathrm{SW}, \mathrm{LA}\}$,$\{\mathrm{SU}\}$)-P $\subseteq \Delta_2^{\mathrm{p}}$.
Lemma U4 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{LC}, \mathrm{SW}, \mathrm{LA}, \mathrm{BS}\}$,$\{\mathrm{EM}\}$)-P $\subseteq$ P.
Lemma U5 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{BX}\}$,$\{\mathrm{ME}\}$)-P $\subseteq$ P.
Lemma U6 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{BX}\}$,$\{\mathrm{SU}\}$)-P $\subseteq \Delta_2^{\mathrm{p}}$.
Lemma U7 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{BR}\}$,$\{\mathrm{ME}\}$)-P $\subseteq \Delta_2^{\mathrm{p}}$.
Lemma U8 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{BR}\}$,$\{\mathrm{SU}\}$)-P $\subseteq \Delta_3^{\mathrm{p}}$.

Let us discuss the results of Theorem 7. It seems that the computational power of the set operations strongly depends on the possession of certain crucial properties. An operation is said to have

- the *block property* if it preserves the property of sets to contain for every $i \in \mathbb{N}$ either all words of length i or no word of length i,
- the *select property* if it can exclude words from a given set (the remaining words being modified or not),
- the *identify property* if it can identify different words. In particular, operations which can replace or cut symbols or subwords are of this type.

The following table shows which properties are possessed by our operations

Operation / Property	UN IN CO	LC SW	LA	EQ BS	BX	BL BR
non-block	$-$	$-$	$+$	$+$	$+$	$+$
select	$-$	$-$	$-$	$-$	$+$	$+$
identify	$-$	$+$	$-$	$+$	$-$	$+$

Now the computational power of the operations can be explained as follows.

- The select property seems to be responsible for the complexity of the emptiness test: If an operation with select property is available, then the emptiness test makes polynomially time-bounded DNA-Pascal as powerful as Δ_2^{p}, otherwise it does not increase the power of DNA-Pascal.
- If only operations with block property are available, then no set test can increase the power of DNA-Pascal. Otherwise the subset test makes polynomial-time bounded DNA-Pascal as powerful as Δ_2^{p}.
- If an operation with non-block and identify property is available, then the membership test (subset test) makes polynomial-time bounded DNA-Pascal as powerful as Δ_2^{p} (Δ_3^{p}, resp.). In the presence of a select operation a simple identify operation has the same effect.

This classification can be a useful tool to obtain hints about the power of other operations not studied in this paper.

5 Lower Bounds

Lemma L1 3-SAT $\in$ DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{LA}\}$,$\{\mathrm{SU}\}$)-P.

PROOF. Let $H(x_1, \ldots, x_n) \equiv \bigwedge_{i=1}^{k} C_i$ be a propositional formula and C_i a clause with three literals. We define the flags $e^1_{ji} =_{\mathrm{df}} \begin{cases} 1, & \text{if } x_j \text{ in } C_i \\ 0 & \text{otherwise} \end{cases}$ and $e^0_{ji} =_{\mathrm{df}} \begin{cases} 1, & \text{if } \overline{x_j} \text{ in } C_i \\ 0 & \text{otherwise} \end{cases}$. Let $a_1, \ldots, a_n \in \{0,1\}$ be the assignments to $x_1, \ldots, x_n$. We construct 'assignment strings' of the form $v(a_1 a_2 \ldots a_n) =_{\mathrm{df}} e^{a_1}_{11} e^{a_1}_{12} \ldots e^{a_1}_{1k} e^{a_2}_{21} e^{a_2}_{22} \ldots e^{a_2}_{2k} \ldots e^{a_n}_{n1} e^{a_n}_{n2} \ldots e^{a_n}_{nk}$. Furtheron we need the set $K_H =_{\mathrm{df}} \{\, v(a_1 a_2 \ldots a_n) \mid a_1, a_2, \ldots, a_n \in \{0,1\} \,\}$ of all possible assignment strings and the set $L_H =_{\mathrm{df}} \bigcup_{i=1}^{k} (\{0,1\}^{i-1} 0 \{0,1\}^{k-i})^n$ of all assignment strings belonging to non-satifying assignments. Now we conclude $H \in$ 3-SAT $\Leftrightarrow K_H \not\subseteq L_H$, and K_H and L_H can easily be produced with the given operations and put into two set variables.

□

Lemma L2 3-SAT $\in$ DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{EQ}\}$,$\{\mathrm{ME}\}$)-P.

PROOF. Let $H(x_1, \ldots, x_n) \equiv \bigwedge_{i=1}^{k} C_i$ be a propositional formula and C_i a clause with three literals. Let $a_1, \ldots .a_n \in \{0,1\}$ be the assignments to $x_1, \ldots, x_n$. We define $u(a_1 \ldots a_n) =_{\mathrm{df}} c_1 \ldots c_k$ where $c_i = 1$ if and only if $(a_1, \ldots, a_n)$ satisfies C_i. One can generate the set $T_H =_{\mathrm{df}} \{\, u(a_1 \ldots a_n) \mid a_1, \ldots, a_n \in \{0,1\} \,\}$ easily by using the operations UN, IN and EQ, and we have $H \in$ 3-SAT $\Leftrightarrow 1^k \in T_H$.

□

Lemma L4 3-$\forall$SAT $\in$ DNA($\{\mathrm{UN}, \mathrm{IN}, \begin{bmatrix} \mathrm{LC} \\ \mathrm{SW} \end{bmatrix}, \mathrm{BX}\}$,$\{\mathrm{SU}\}$)-P.

PROOF. The LC case: Let $H(x_1, \ldots, x_n, y_1, \ldots, y_m) \equiv \bigwedge_{i=1}^{k} C_i$ be a propositional formula and C_i a clause with three literals each. The question $H \in$ 3-$\forall$SAT can be decided by an DNA-Pascal program doing the following:

1. The set $T^1_H =_{\mathrm{df}} \{\, a_1 \ldots a_n\, b_1 \ldots b_m \in \{0,1\}^{n+m} \mid H(a_1, \ldots, a_n, b_1, \ldots, b_m) = 1 \,\}$ is constructed as in the proof of Theorem 3.
2. The set $T^2_H =_{\mathrm{df}} \{\, b_1 \ldots b_m \mid \exists a_1 \ldots \exists a_n \ (H(a_1, \ldots, a_n, b_1, \ldots, b_m) = 1) \,\}$ is constructed by applying LC n times to T^1_H.
3. H is accepted if and only if $\{0,1\}^m \subseteq T^2_H$.

The SW case: Again let $H(x_1, \ldots, x_n, y_1, \ldots, y_m) \equiv \bigwedge_{i=1}^{k} C_i$ be a propositional formula and C_i a clause with three literals each. To decide the question $H \in$ 3-$\forall$SAT we want to follow the ideas of the proof of the LC case, but we have to face the problem that by dividing the strings in substrings we do not know anymore where the substring $b_1 \ldots b_m$ starts. To overcome this we introduce an additional structure to the assignment strings: instead of strings $a_1 a_2 \ldots a_l$ with $a_1, a_2, \ldots, a_l \in \{0,1\}$ we use new strings $w(a_1 a_2 \ldots a_l) =_{\mathrm{df}} 0 a_1 0 a_2 0 \ldots 0 a_l 0$. Now we can do the following:

1. The sets $T_H^1 =_{\rm df} \{\, w(a_1 \ldots a_m)\, 11\, w(b_1 \ldots b_n) \mid H(a_1, \ldots, a_n, b_1, \ldots, b_m) = 1 \,\}$ and $T_H^2 =_{\rm df} \{\, 11\, w(b_1 \ldots b_n) \mid b_1, \ldots, b_n \in \{0,1\} \,\}$ are constructed starting with IN and then using BX and UN.

2. We construct the set $T_H^3 := \mathrm{Sw}(T_H^1)$. Notice that $T_H^2 \cap T_H^3 = \{\, 11\, w(b_1 \ldots b_n) \mid \exists a_1 \ldots \exists a_m\, (H(a_1, \ldots, a_n, b_1, \ldots, b_m) = 1) \,\}$.

3. H is accepted if and only if $T_H^2 \subseteq T_H^3$.

□

Lemma L5 3-∀SAT ∈ DNA({UN, IN, EQ},{SU})-P.

PROOF. Let $H(x_1, \ldots, x_n, y_1, \ldots, y_m) \equiv \bigwedge_{i=1}^{k} C_i$ be a propositional formula and C_i a clause with three literals each. We generate the set $T_H =_{\rm df} \{\, u(a_1 \ldots a_n b_1 \ldots b_m)\, b_1 \ldots b_m \mid a_1, \ldots, a_n, b_1, \ldots, b_m \in \{0,1\} \,\}$ as in the proof of Lemma L2. Next we generate $T'_H =_{\rm df} 1^k\{0,1\}^m$, and we conclude $H \in$ 3-∀SAT $\Leftrightarrow T'_H \subseteq T_H$.

6 Upper Bounds

Lemma U1 DNA({UN, IN, CO, LC, SW},{ME, SU})-P ⊆ P.

PROOF. The exclusive use of the operations UN, IN, CO, LC, and SW has the consequence that the sets produced in the set variables contain for every $i \in \mathbb{N}$ either every word of length i or no word of length i. Hence, every set variable can be replaced by an array storing the word lengths occuring in this variable. Every test can easily be simulated by consulting these arrays.

□

Lemma U2 DNA({UN, IN, CO, LC, SW, LA},{ME})-P ⊆ P.

PROOF. Let L be a language decidable by a ({UN,IN,CO,LC,SW,LA},{ME})-DNA-Pascal program P in time p where p is a polynomial. Let $T_1, \ldots, T_k$ be the set variables in that program. As in the proof of Theorem 4 we define the set

$$R_P =_{\rm df} \{(x, a_1 \ldots a_m, z, i, t) \mid \begin{array}{l} z \in T_i \text{ at step } t \text{ of program } P \text{ on input } x \text{ and} \\ \text{there have been } m \text{ tests before step } t \text{ and} \\ a_1, \ldots, a_m \in \{0,1\} \text{ are the results of these tests}\}. \end{array}$$

Also in this more general case the set R_P is in P. However, the argument is a bit more sophisticated than in the case of Lipton's model where the strings in the set variables are never modified during the computation. In the present case the strings can be concatenated, devided, cut, and extended. To decide $(x, a_1 \ldots a_m, z, i, t) \in R_P$ it is now not only sufficient to study the membership history of $z \in T_j$ for all $j \in \{1, \ldots, k\}$ during the corresponding run of P on x but also this history for all subwords of z and their possible extensions. Define $Q(z) =_{\rm df} \{\, y \mid \exists y_1, y_2\ (z = y_1 y y_2) \,\}$ as the set of all subwords of z. We introduce three-dimensional arrays $s_1, \ldots, s_k$ with the intention that for all $y \in Q(z)$ $s_j[y, r, s]$ should be 1 if and only if $\{0,1\}^r y \{0,1\}^s \cap I(T_j) \neq \emptyset$. Now we replace the set instructions in P by Pascal instructions without set variables. Without loss of generality we assume that for every instruction $T_j := T_u \cdot T_v$ in P, it holds that $j \neq u$ and $j \neq v$. Here is the table of replacements:

set instruction	replaced by Pascal instruction: for ($y \in Q(z)$ and $r, s \in \{0, 1, \ldots, p(n)$) do
$T_j := T_u \cup T_v$	$s_j[y, r, s] := \max(s_u[y, r, s], s_v[y, r, s])$
$T_j := \mathrm{In}(n)$	if $\|y\| + r + s = n$ then $s_j[y, r, s] := 1$
$T_j := T_u \cdot T_v$	$s_j[y, r, s] := \max_{y_1 y_2 = y} s_u[y_1, r, 0] \cdot s_v[y_2, 0, s]$
$T_j := \backslash T_u$	$s_j[y, r, s] := s_u[y, r + 1, s]$
$T_j := \mathrm{Sw}(T_u)$	$s_j[y, r, s] := \max_{\substack{r \le r' \le p(n) \\ s \le s' \le p(n)}} s_u[y, r', s']$
$T_j := a \cdot T_u$	begin if $ay \in Q(z)$ then $s_j[ay, 0, s] := s_u[y, 0, s]$; $s_j[y, r + 1, s] := s_u[y, r, s]$ end

Now let this modified program run on input x where the r-th test is replaced by the test "$a_r = 1$" ($r = 1, \ldots, m$). Finally, after having simulated t steps of P in such a way the answer to $(x, a_1 \ldots a_m, z, i, t) \in R_P$ is the value of $s_i[z, 0, 0]$ at this moment.

The simulation of the DNA-Pascal program P on input x by a Pascal program with oracle is exactly as in the proof of Theorem 4, where a membership test $z \in T_i$ (in step t with exactly m tests before that have the results $a_1, \ldots, a_m$) can be replaced by the oracle query $(x, a_1 \ldots a_m, z, i, t) \in R_P$ which is a query to a P-oracle. Thus $L \in \mathrm{P}^{\mathrm{P}} = \mathrm{P}$.

□

Lemma U3 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{LC}, \mathrm{SW}, \mathrm{LA}\}$,$\{\mathrm{SU}\}$)-P $\subseteq \Delta_2^{\mathrm{p}}$.

PROOF. This case is very similar to the U2 case. The difference is only that subset tests instead of membership tests have to be replaced by oracle queries. Such a subset test $T_i \subseteq T_j$ in step t can be replaced by the oracle query

$$\forall z \, ((\, |z| \le p(|x|) \wedge (x, a_1 \ldots a_m, z, i, t) \in R_P) \Rightarrow (x, a_1 \ldots a_m, z, j, t) \in R_P),$$

assumed that there were exactly m tests before with the results $a_1, \ldots, a_m$. However, this is a query to a coNP-oracle. Thus $L \in \mathrm{P}^{\mathrm{coNP}} = \Delta_2^{\mathrm{p}}$.

□

Lemma U4 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{LC}, \mathrm{SW}, \mathrm{LA}, \mathrm{BS}\}$,$\{\mathrm{EM}\}$)-P $\subseteq$ P.

PROOF. Let L be a language decidable by a ({UN,IN,CO,LC,SW,LA,EQ},{EM})-DNA-Pascal program P in polynomial time. Let $T_1, \ldots, T_k$ be the set variables in that program. To replace an emptiness test in P we just have to know at any time of the computation whether the set variables are empty or not. Therefore we introduce variables $s_1, \ldots, s_k$ with the intention that $s_i = 0$ if and only if T_i is empty. Set instructions in P are replaced by Pascal instructions without set variables. Here every initialization $T_j := \mathrm{In}(n)$ is simulated by $s_j := 1$. Union $T_j := T_u \cup T_v$ and concatenation $T_j := T_u \cdot T_v$ are replaced by $s_j := \max(s_u, s_v)$ and $s_j := \min(s_u, s_v)$, resp. In the case of left cut $T_j := \backslash T_u$, subword building $T_j := \mathrm{Sw}(T_u)$, left adding $T_j := a \cdot T_u$, and bit swap $T_j := \mathrm{Bs}(T_u, m, a)$, the "emptiness flag" of T_u is inherited to T_j, therefore $s_j := s_u$. Now an emptiness test $T_i = \emptyset$ at any time of the program has the same answer as the test "$s_i = 0$" at that time. Clearly, such a modifyed program runs in polynomial time.

□

Lemma U5 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{BX}\}$,$\{\mathrm{ME}\}$)-P $\subseteq$ P.

Lemma U6 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{BX}\}$,$\{\mathrm{SU}\}$)-P $\subseteq \Delta_2^{\mathrm{p}}$.

PROOF. The proofs of the Lemmas U5 and U6 follow exactly the argumentation of the proofs of Lemmas U2 and U3. The case is a bit easier here, as the words in the set variables can only get longer during the computation. Thus we do not have to study extensions of which we are assumed to be cut off later. Therefore it is sufficient to introduce one-dimensional arrays $s_1, \ldots, s_k$ with the intention that $s_j[y]$ should be 1 if $y \in T_j$ and 0 otherwise for all $y \in Q(z)$. Now we replace the set instructions in P by Pascal instructions without set variables. Without loss of generality we assume that for every instruction $T_j := T_u \cdot T_v$ in P, it holds that $j \neq u$ and $j \neq v$. Here is the table of replacements:

set instruction	replaced by Pascal instruction: for $y \in Q(z)$ do
$T_j := T_u \cup T_v$	$s_j[y] := \max(s_u[y], s_v[y])$
$T_j := \mathrm{In}(n)$	if $\|y\| = n$ then $s_j[y] := 1$
$T_j := T_u \cdot T_v$	$s_j[y] := \max_{y_1 y_2 = y} s_u[y_1] \cdot s_v[y_2]$
$T_j := \mathrm{Bx}(T_u, m, a)$	if $y(m) = a$ then $s_j[y] := s_u[y]$ else $s_j[y] := 0$

□

Lemma U7 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{BR}\}$,$\{\mathrm{ME}\}$)-P $\subseteq \Delta_2^{\mathrm{p}}$.

PROOF. Let L be a language decidable by a ({UN,IN,CO,BR},{ME})-DNA-Pascal program P in time p where p is a polynomial. Let $T_1, \ldots, T_k$ be the set variables in that program. As in the proofs of the preceeding Lemmas and Lemma 4 we will replace membership tests by queries to the oracle set R_P. We do not know whether R_P is in P, because the BR operation does not allow an easy study of the history of a word. Our trick now is not to deal with words but with expressions describing possible histories of the words.

These expressions are defined as follows. The initial creation of a word $b_1 b_2 \ldots b_m$ by $T_j := \mathrm{In}(m)$ is described by the expression $H = b_1 b_2 \ldots b_m$. Now let the expressions H_1 and H_2 describe the possible history of two words z_1 in T_u and z_2 in T_v, resp., at a given moment of the computation. Then after the execution of $T_j := T_u \cdot T_v$ the expression $(H_1)(H_2)$ describes a possible history of the word $z_1 z_2$ in T_j, and after the execution of $T_j := \mathrm{Br}(T_u, m, a, y)$ the expression $(H_1)^y_{m,a}$ describes a possible history of the word vyw in T_j, if $z_1 = vaw$ and $|v| = m - 1$. If $z_1(m) \neq a$ then $(H_1)^y_{m,a}$ is not the history of a word. If H describes the history of a word then define $\omega(H)$ as this word and define

$$R'_P =_{\mathrm{df}} \{(x, a_1 \ldots a_m, H, i, t) \mid \begin{array}{l} \omega(H) \in T_i \text{ via the history decribed by } H \\ \text{at step } t \text{ of program } P \text{ on input } x; \\ \text{there have been } m \text{ tests before step } t \text{ and} \\ a_1, \ldots, a_m \in \{0, 1\} \text{ are the results of these tests}\}. \end{array}$$

Obviously ω is a polynomial-time computable function. That R'_P is in P can be seen as follows. Define $Q'(H)$ as the set of all subexpressions of H. Now we modify the program P by introducing one-dimensional arrays $s_1, \ldots, s_k$ with the intention that $s_j[H']$ should be 1 if $\omega(H') \in T_j$ via the history described by H' and 0 otherwise, for all $H' \in Q'(H)$. We achieve this by replacing the set instructions in P by Pascal instructions without set variables. Here is the table of replacements:

set instruction	replaced by Pascal instruction for $H' \in Q'(H)$ do
$T_j := T_u \cup T_v$	$s_j[H'] := \max(s_u[H'], s_v[H'])$
$T_j := \mathrm{In}(n)$	if $H' \in \{0,1\}^n$ then $s_j[H'] := 1$ else $s_j[H'] := 0$
$T_j := T_u \cdot T_v$	if $H' = (H_1)(H_2)$ then $s_j[H'] := s_u[H_1] \cdot s_v[H_2]$ else $s_j[H'] := 0$
$T_j := \mathrm{Br}(T_u, m, a, y)$	if $(H' = (H_1)^y_{m,a})$ and $((\omega(H_1))(m) = a)$ then $s_j[H'] := s_u[H_1]$ else $s_j[H'] := 0$

To decide $(x, a_1 \dots a_m, H, i, t) \in R'_P$ let the modified program run on x where the r-th test is replaced by the test "$a_r = 1$" $(r = 1, \dots, m)$. Finally, after the simulation of t steps of P in such a way the answer to $(x, a_1 \dots a_m, H, i, t) \in R'_P$ is the value of $s_i[H]$ at this moment. Now observing

$$(x, a_1 \dots a_m, z, i, t) \in R_P \Leftrightarrow \exists H \ (\ |H| \leq p(|x|) \wedge \omega(H) = z \wedge (x, a_1 \dots a_m, H, i, t) \in R'_P)$$

we obtain $R_P \in$ NP.

The simulation of the DNA-Pascal program P on input x by an ordinary Pascal program with oracle is as in the proof of Theorem 4, where the replacement of the membership test $z \in T_i$ in step t by the oracle query $(x, a_1 \dots a_m, z, i, t) \in R_P$ yields a P^{NP}-computation.

□

Lemma U8 DNA($\{\mathrm{UN}, \mathrm{IN}, \mathrm{CO}, \mathrm{BR}\}$,$\{\mathrm{SU}\}$)-P $\subseteq \Delta^{\mathrm{p}}_3$.

PROOF. This proof is the same as the proof of Lemma U7 with the only difference that subset tests have to be replaced by oracle queries. Such a test $T_i \subseteq T_j$ (in step t with exactly m tests before that have the results $a_1, \dots, a_m$) can be replaced by the oracle query

$$\forall z \ ((\ |z| \leq p(|x|) \wedge (x, a_1 \dots a_m, z, i, t) \in R_P) \Rightarrow (x, a_1 \dots a_m, z, j, t) \in R_P),$$

which is a query to a Π^{p}_2-oracle and leads to a $\mathrm{P}^{\Pi^{\mathrm{p}}_2}$-computation.

□

Acknowledgement

We are grateful to an anonymous referee for a lot of very valuable hints, and to Heribert Vollmer and Gundula Niemann for interesting discussions.

References

[Adl94] L. M. ADLEMAN. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[BC94] D. BOVET AND P. CRESCENZI. *Introduction to the Theory of Complexity*. Prentice Hall, New York, London, Toronto, Sydney, Tokyo, Singapore, 1994.

[BV93] E. BERNSTEIN AND U. VAZIRANI. Quantum complexity theory. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pages 11–20, 1993.

[Deu85] D. DEUTSCH. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society London, Vol. A400*, pages 97–117, 1985.

[Lip94] R. J. LIPTON. Speeding up computations via molecular biology. Technical report, Princeton University, 1994.

[Lip95] R. J. LIPTON. Using DNA to solve NP-complete problems. Technical report, Princeton University, 1995.

[Pud94] P. PUDLÁK. Complexity theory and genetics. In *Proceedings of 9th Conference on Structure in Complexity Theory*, pages 183–195, 1994.

[Sho94] P. W. SHOR. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.