On the Power of DNA-Computers

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 are investigated, and it is proved that their computational power in polynomial time can be characterized by one of the complexity classes P, Δ_2^{p}, Δ_3^{p} or even PSPACE. Restricting Liptons model to DNA strings of logarithmic length one can compute exactly the problems in L.

1 Introduction

In the last 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 bitwise by electric and electromagnetic means, and the information is modified by using just these properties of the memory. 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 computers. 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 use biological properties of DNA strings to store and modify information. The general idea is to use a large number of DNA strings as "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 not to investigate the possibilities of "real biological computing", but to develop Adleman's and Lipton's ideas as an interesting new theoretical approach to parallel computing by manipulating large sets of strings. Therefore we introduce in Section 2 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].) That various other operations which do not decrease the length of the words in the test tubes have the same power as Lipton's model is shown in Section 4. The situation changes (Section 5) if we allow length-decreasing operations. Here the power of the model strongly depends on the types of set tests allowed. Whereas in the case of membership test and emptiness test the class of problems solvable in polynomial time is proved to coincide with Δ_2^{p}, in the case of subset test the power of the model can be characterized by Δ_3^{p}. In Section 6 we prove that the additional use of the operation *intersection* increases the power of the model to PSPACE, what is everything we can expect from parallel polynomially time-bounded computations. We show (Section 7) that the models observed in the last sections need exponentially many strings in the set variables to derive the full power of their computations. Finally, we investigate another kind of resource bound in Section 8: What happens to DNA-Pascal, if all variables (including every string contained in a set variable) are restricted to logarithmic length? Things turn out to have a simpler structure than in the polynomially time-bounded case. Now already the collection of operations and tests that Lipton uses, gives logarithmically space-bounded DNA-Pascal programs the whole power of P.

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 operations and tests defined on these variables. By one of the usual one-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. Let m be a variable for natural numbers, let x be a word variable and let $T,\ T_1$ and T_2 be set variables. The different types of DNA-computers we shall consider use the following set operations and set tests:

Operations

(UN)	Union	$T := T_1 \cup T_2$
(EX)	Extraction	stands for both, BX and SX; i. e. all results stated below are valid for BX as well as for SX
(BX)	Bit Extraction	$T := \mathrm{Bx}(T_1, m, a)$ where $\mathrm{Bx}(T_1, m, a) =_{\mathrm{df}} T_1 \cap (\{0,1\}^{m-1} a \{0,1\}^*)$, $a \in \{0,1\}$ and $m \geq 1$
(SX)	Subword Extraction	$T := \mathrm{Sx}(T_1, x)$ where $\mathrm{Sx}(T_1, x) =_{\mathrm{df}} T_1 \cap (\{0,1\}^* x \{0,1\}^*)$ and $x \in \{0,1\}^*$
(IN)	Initialization	$T := \mathrm{In}(m)$ where $\mathrm{In}(m) =_{\mathrm{df}} \{0,1\}^m$ and $m \geq 0$
(EW)	Empty Word	$T := \{\varepsilon\}$
(RA)	Right Adding	$T := T_1 \cdot a$ where $T_1 \cdot a =_{\mathrm{df}} \{\, za \mid z \in T_1 \,\}$
(LA)	Left Adding	$T := a \cdot T_1$ where $a \cdot T_1 =_{\mathrm{df}} \{\, az \mid z \in T_1 \,\}$
(CO)	Concatenation	$T := T_1 \cdot T_2$
(RC)	Right Cut	$T := T_1/$ where $T_1/ =_{\mathrm{df}} \{\, z/ \mid z \in T_1 \,\}$ and $za/ =_{\mathrm{df}} z$ for $a \in \{0,1\}$ and $\varepsilon/ =_{\mathrm{df}} \varepsilon$
(LC)	Left Cut	$T := \backslash T_1$ where $\backslash T_1 =_{\mathrm{df}} \{\, \backslash z \mid z \in T_1 \,\}$ $\backslash az =_{\mathrm{df}} z$ for $a \in \{0,1\}$ and $\backslash\varepsilon =_{\mathrm{df}} \varepsilon$
(IS)	Intersection	$T := T_1 \cap T_2$

Conditions

(SU)	Subset Test	$T_1 \subseteq T_2$
(EM)	Emptiness Test	$T = \emptyset$
(ME)	Membership Test	$x \in T$

To the time complexity of such programs: The execution of each of these operations and tests takes one step with the exclusion of the operations IN and CO. 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 $T := \mathrm{In}(m)$ or $T := T_1 \cdot T_2$ 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 stays 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]). This is reflected by the fact that a set variable can be used twice (or more) without destroying its contents.

In the next sections we are interested in DNA-Pascal programs that run in polynomial time using 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 tests from $\mathcal{T}$.

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

Propositon 2 $\mathrm{P}^{\mathrm{DNA}(\mathcal{O},\mathcal{T})\text{-P}} = \mathrm{DNA}(\mathcal{O},\mathcal{T})$-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 considers 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):

- **extract operation** (corresponds to our operation BX),
- **detect operation** (corresponds to our test EM),
- **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),
- **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.

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 their 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,BX,IN},{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,BX,IN},{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 variables 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
  $T := \mathrm{In}(n)$; (put all possible "assignment strings" in T)
  for $i := 0$ to k do
    begin
      $T_1 := \mathrm{Bx}(T, \sigma(i,1), a_{i,1})$; (select all strings satisfying $x_{\sigma(i,1)}^{a_{i,1}}$)
      $T_2 := \mathrm{Bx}(T, \sigma(i,2), a_{i,2})$; (select all strings satisfying $x_{\sigma(i,2)}^{a_{i,2}}$)
      $T_3 := \mathrm{Bx}(T, \sigma(i,3), a_{i,3})$; (select all strings satisfying $x_{\sigma(i,3)}^{a_{i,3}}$)
      $T := T_1 \cup T_2$; (collect all strings
      $T := T \cup T_3$ satisfying the i-th clause)
    end;
  if $T = \emptyset$ then reject else accept (only assignments which satisfy 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,BX,IN},{EM})-P $= \Delta_2^{\mathrm{p}}$.

PROOF. [$\supseteq$]: This follows from a chain of inclusions using Proposition 2 and Theorem 3:

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

[$\subseteq$]: Let L be a language decidable by a ({UN,BX,IN},{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

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

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 use basically 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 $z \in T_j$ 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 $z(s)$ be the s-th symbol in the word z):

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

The r-th test ($r = 1, \ldots, m$) is replaced by a_r. 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,BX,IN$\}$,$\{$EM$\}$)-DNA-Pascal program P by an ordinary Pascal program P' using an NP-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 $t+1$ 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 an NP-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 polynomial time-bounded Pascal program without set variables that asks queries to an coNP-oracle.

□

4 More operations that are not length-decreasing

After having estimated the power of Lipton's model of DNA-computing we will study extended models having other set operations besides UN, EX, and IN and other tests besides EM. In this section we consider set operations which do not decrease the length of words in a set. From the list of set operations described in Section 2 these are subword extraction (SX), empty word initialization (EW), right adding (RA), left adding (LA), and concatenation (CO).

Since we want to have at least the abilities of Lipton's model we will include UN and BX always in the set operations. However, using the operation subword extraction (SX) instead of BX we would obtain the same results in what follows. To see this we first prove that SX can be replaced by UN and BX.

Lemma 5 DNA($\mathcal{O} \cup \{\mathrm{SX}\}$,$\mathcal{T}$)-P $\subseteq$ DNA($\mathcal{O} \cup \{\mathrm{UN}, \mathrm{BX}\}$,$\mathcal{T}$)-P *for all collections $\mathcal{O}$ of set operations and all collections $\mathcal{T}$ of set tests.*

PROOF. Let P be a DNA-Pascal program using operation SX, and let p be a polynomial bounding the running time of P. If P uses the statement $T := \mathrm{Sx}(T_1, a_1 a_2 \ldots a_m)$ with $a_1, a_2, \ldots, a_m \in \{0, 1\}$ during its work on input x then this can be replaced equivalently by the following statement including only the operations UN and BX and using a new set variable T_2 not used in P.

```
begin
  T := Bx(T, 1, 0);
  T := Bx(T, 1, 1);                    (initialize T with ∅)
  for i := 0 to p(|x|) do
    begin
      T_2 := T_1;
      for j := 1 to m do               (results in
        T_2 := Bx(T_2, i + j, a_j);    T := T ∪ (T_1 ∩ ({0,1}^i a_1 a_2 ... a_m {0,1}^*)))
```

$T := T \cup T_2$;
end
end

□

Now knowing that in presence of the operation UN the operation BX is at least as strong as the operation SX we prove in the following upper bounds with BX and lower bounds with SX. Thus, all statements formulated below with the operation EX are valid for BX as well as for SX.

To be able to fill the initially empty set variables with words we need either the operation IN, or the operations EW and RA (respectively EW and LA). Relations between the power of {IN}, {EW,RA}, and {EW,LA} are given in the following lemma.

Lemma 6 *For every collection $\mathcal{O}$ of set operations including* UN *and* EX*, and for every collection $\mathcal{T}$ of set tests*

1. DNA($\mathcal{O} \cup \{\mathrm{IN}\}, \mathcal{T}$)-P $\overset{(a)}{\subseteq}$ DNA($\mathcal{O} \cup \{\mathrm{EW}, \mathrm{RA}\}, \mathcal{T}$)-P $\overset{(b)}{\subseteq}$ DNA($\mathcal{O} \cup \{\mathrm{IN}, \mathrm{CO}\}, \mathcal{T}$)-P,
2. DNA($\mathcal{O} \cup \{\mathrm{IN}\}, \mathcal{T}$)-P $\overset{(a)}{\subseteq}$ DNA($\mathcal{O} \cup \{\mathrm{EW}, \mathrm{LA}\}, \mathcal{T}$)-P $\overset{(b)}{\subseteq}$ DNA($\mathcal{O} \cup \{\mathrm{IN}, \mathrm{CO}\}, \mathcal{T}$)-P.

PROOF. [1.(a)]: The instruction $T := \mathrm{In}(m)$ can be replaced equivalently by the instruction

begin
  $T := \{\varepsilon\}$;
  for $i := 1$ to m do
    begin $T' := T \cdot 0; T := T \cdot 1; T := T \cup T'$ end
end

where T' is a new set variable not used in the program before.

[1.(b)]: The instruction $T := \{\varepsilon\}$ can be replaced equivalently by the instruction $T := \mathrm{In}(0)$ whereas the instruction $T := T' \cdot a$ can be replaced equivalently by

$$\text{begin } T := \mathrm{In}(1);\ T := \mathrm{Bx}(T, 1, a);\ T := T' \cdot T \text{ end.}$$

[2.]: Statement 2 is proven in the same way.

□

Now let us compare different tests. Besides the emptiness test (EM) we also consider the membership test (ME) and the subset test (SU). Membership tests are weaker than emptiness tests (cf. Theorem 8.2) whereas subset tests are stronger in many important cases as shown in the following lemma.

Lemma 7 *1.* DNA($\mathcal{O}$,{EM})-P $\subseteq$ DNA($\mathcal{O}$,{SU})-P.

2. *If* $\mathrm{EX}, \mathrm{IN} \in \mathcal{O}$ *or* $\mathrm{RA}, \mathrm{EW} \in \mathcal{O}$ *or* $\mathrm{LA}, \mathrm{EW} \in \mathcal{O}$*, then*

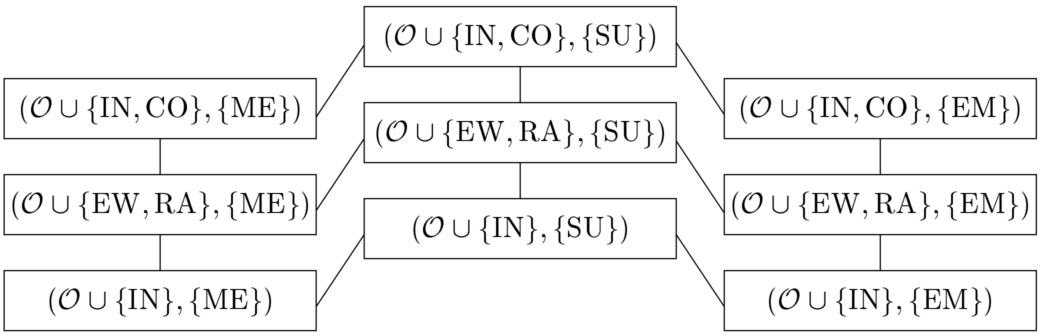
$$\text{DNA}(\mathcal{O},\{\text{ME}\})\text{-P} \subseteq \text{DNA}(\mathcal{O},\{\text{SU}\})\text{-P}.$$

PROOF. [1.]: Let P be an ($\mathcal{O}$,{EM})-DNA-Pascal program, and let T' be a set variable that does not occur in P. Notice that T' is always empty as it is not used in P. So we can replace any emptiness test $T = \emptyset$ by the subset test $T \subseteq T'$.

[2.]: Let P be an ($\mathcal{O}$,{ME})-DNA-Pascal program, and let T' be a set variable that does not occur in P. To replace the membership test $x \in T$ we have to "fill" T' with x and make the subset test $T' \subseteq T$. Filling T' with x can be done by initializing T' with $\mathrm{In}(|x|)$ and repeated extraction according to the characters of x, or by initializing T' with $\{\varepsilon\}$ and repeated adding.

□

The relationships between the powers of different collections of set operations and set tests given by the Lemmas 6 and 7 are illustrated in the following figure (where we restrict ourselves to RA, the case LA is analogous). Let $\mathcal{O} = \{\mathrm{UN}, \mathrm{EX}\}$.



Note that in the upper slice {IN,CO} can be replaced equivalently by {IN,EW,RA,CO} and in the middle slice {EW,RA} can be replaced equivalently by {IN,EW,RA}. In the middle row {SU} can be replaced equivalently by {ME,EM,SU}.

The following theorem gives upper bounds for the power of polynomial time computations with the various set operations and set tests.

Theorem 8 *1.* DNA({UN,EX,IN,EW,RA,LA,CO},{ME,EM,SU})-P $\subseteq \Delta_2^{\mathrm{p}}$,

2. DNA({UN,EX,IN,EW,RA,LA,CO},{ME})-P $\subseteq$ P.

PROOF. [1.]: Following Lemma 6 and Lemma 7 we allow only set operations UN, BX, IN and CO, and subset tests. Let L be a language decidable by a ({UN,BX,IN,CO},{SU})-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

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

Also in this more general case the set R_P is in P. However, the argument is a bit more sophisticated since contrary to the case of Lipton's model the strings in the set variables can be concatenated by the set operation CO. To decide $(x, a_1 \dots a_m, z, i, t) \in R_P$ it is now not only sufficient to study the membership history of $z \in T_j$ for all $j \in \{1, \dots, k\}$ during the corresponding run of P on x but also this history for all subwords of z. Define $Q(z) =_{\mathrm{df}} \{\, y \mid \exists y_1, y_2 \in \{0,1\}^* \; (z = y_1 y y_2) \,\}$ as the set of all subwords of z. We introduce one-dimensional arrays $s_1, \dots, s_k$ with the intention that $s_j[y]$ should be 1 if $y \in T_j$ and 0 otherwise for all $y \in Q(z)$. This is achieved by replacing 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
$T_j := \mathrm{In}(n)$	for $y \in Q(z)$ do if $\lvert y \rvert = n$ then $s_j[y] := 1$
$T_j := T_u \cup T_v$	for $y \in Q(z)$ do $s_j[y] := \max(s_u[y], s_v[y])$
$T_j := \mathrm{Bx}(T_u, m, a)$	for $y \in Q(z)$ do if $y(m) = a$ then $s_j[y] := s_u[y]$ else $s_j[y] := 0$

$T_j := T_u \cdot T_v$	for $y \in Q(z)$ do
	begin
	  $s_j[y] := 0$;
	  for $y_1 \in Q(z)$ do
	    for $y_2 \in Q(z)$ do
	      if $y = y_1 y_2$ then $s_j[y] := \max(s_j[y], s_u[y_1] \cdot s_v[y_2])$
	end

Now let this modified program run on input x where the r-th test is replaced by a_r $(r = 1, \dots, m)$. Finally, after having simulated t steps of P in such a way the answer to $(x, a_1 \dots a_m, z, i, t) \in R_P$ is the value of $s_i[z]$ 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 subset test $T_i \subseteq T_j$ in step t can be replaced by the oracle query

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

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

[2.]: The proof of the second case follows the same ideas as the proof of the first case. The difference is that only membership tests instead of subset tests have to be replaced by oralcle queries. Such a test $z \in T_i$ (in step t with exactly m tests before that have the results $a_1, \dots, a_m$) can be replaced by the oracle query $(x, a_1 \dots a_m, z, j, t) \in R_P$ which is a query to a P-oracle. Thus $L \in \mathrm{P}^{\mathrm{P}} = \mathrm{P}$.

□

Theorem 9

1. DNA($\mathcal{O}$,{ME})-P $=$ P *for every collection* $\mathcal{O} \subseteq \{\mathrm{UN}, \mathrm{EX}, \mathrm{IN}, \mathrm{EW}, \mathrm{RA}, \mathrm{LA}, \mathrm{CO}\}$,

2. DNA({UN,EX,IN,EW,RA,LA,CO},{ME,EM,SU})-P $= \Delta_2^{\mathrm{P}}$,

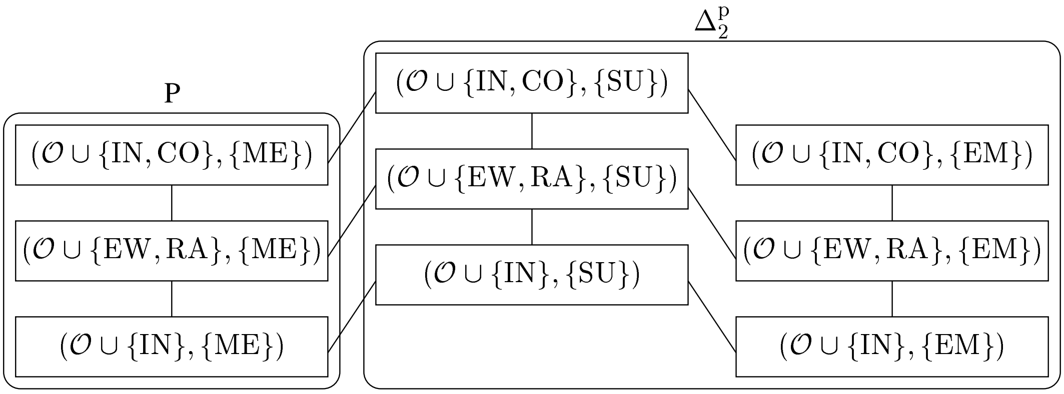
3. DNA({UN,EX,IN},{EM})-P $= \Delta_2^{\mathrm{p}}$.

PROOF. Statement 1 is an immediate consequence of Theorem 8.2. Because of Lemma 6, Lemma 7, and Theorem 8.1 the proof of the other statements reduces to the proof of $\Delta_2^{\mathrm{p}} \subseteq$ DNA({UN,SX,IN},{EM})-P (see discussion after Lemma 5). As in the proof of Theorem 4 it is sufficient to prove that 3-SAT $\in$ DNA({UN,SX,IN},{EM})-P. We will do that by modifying Lipton's proof of 3-SAT $\in$ DNA({UN,BX,IN},{EM})-P (see Theorem 3). The latter proof uses strings $a_1 a_2 \dots a_n$ representing assignments to the variables $x_1, x_2, \dots, x_n$ and translates conditions to the variables x_j to conditions to the bit a_j using $\mathrm{Bx}(\cdot, j, \cdot)$, i. e. operation BX. The main difficulty when adapting this proof is that the j-th bit a_j cannot be addressed directly by the operation SX. To overcome this we introduce an additional structure to the assignment strings: instead of assignment strings $a_1 a_2 \dots a_n$ with $a_1, a_2, \dots, a_n \in \{0, 1\}$ we use new assignment strings $w(a_1 a_2 \dots a_n) =_{\mathrm{df}} 10^2 1\, a_1\, 10^3 1\, a_2\, 10^4 1\, a_3\, \dots 10^{n+1} 1\, a_n\, 10^{n+2} 1$ again with $a_1, a_2, \dots, a_n \in \{0, 1\}$. Addressing the j-th bit in $a_1 a_2 \dots a_n$ is now the same as looking for the subword $10^{j+1} 1\, a_j$ in $w(a_1 a_2 \dots a_n)$. Note that the length of the word $w(a_1 a_2 \dots a_n)$ is $\frac{1}{2}(n^2 + 11n + 8)$. Now the DNA-Pascal program in the proof of Theorem 3 is modified along these lines as follows.

begin
  $T := \mathrm{In}(\frac{1}{2}(n^2 + 11n + 8))$;
  for $j := 1$ to n do
    begin
      $T_1 := \mathrm{Sx}(T, 10^{j+1} 1\, 0\, 10^{j+2} 1)$;   (results in
      $T_2 := \mathrm{Sx}(T, 10^{j+1} 1\, 1\, 10^{j+2} 1)$; $T := \{\, w(a_1 a_2 \dots a_n) \mid a_1, a_2, \dots, a_n \in \{0, 1\} \,\}$)
      $T := T_1 \cup T_2$
    end
  for $i := 1$ to k do
    begin
      $T_1 := \mathrm{Sx}(T, 10^{\sigma(i,1)+1} 1\, a_{i,1})$;   (results in
      $T_2 := \mathrm{Sx}(T, 10^{\sigma(i,2)+1} 1\, a_{i,2})$; $T := T \cap \{w(a_1 a_2 \dots a_n) \mid a_1 a_2 \dots a_n$ satisfies
      $T_3 := \mathrm{Sx}(T, 10^{\sigma(i,3)+1} 1\, a_{i,3})$;   the i-th clause$\}$)
      $T := T_1 \cup T_2$;
      $T := T \cup T_3$
    end
  if $T = \emptyset$ then accept else reject
end

□

The inclusional relationships given by the Lemmas 6 and 7 and the results of Theorem 9 are represented by the following figure. Again the LA case is left out, as it is analogous to the RA case. Let $\mathcal{O} = \{\mathrm{UN}, \mathrm{EX}\}$.



5 Length-decreasing operations

In this section we study the influence of the length-decreasing set operations RC and LC. At first we state that in the presence of one (or both) of these operations the membership test is at least as powerful as the emptiness test.

Lemma 10 *If $\mathcal{O}$ is a collection of operations such that* $\mathrm{RC} \in \mathcal{O}$ *or* $\mathrm{LC} \in \mathcal{O}$ *then*

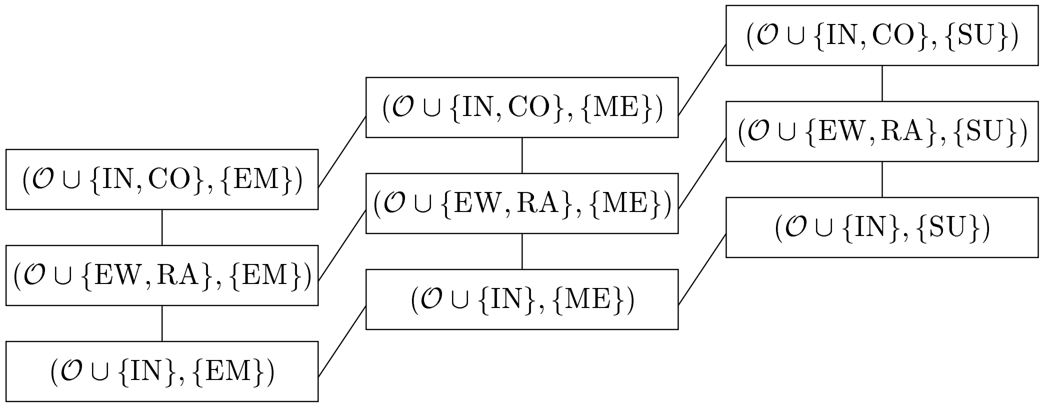
$$\text{DNA}(\mathcal{O},\{\text{EM}\})\text{-P} \subseteq \text{DNA}(\mathcal{O},\{\text{ME}\})\text{-P}.$$

PROOF. Let P be a ($\mathcal{O}$,{EM})-DNA-Pascal program, and let every set variable T be length bounded by the polynomial p. We can replace each emptiness test $T = \emptyset$ by a condition $s(T) = 0$ where $s(T)$ is the ($\mathcal{O}$,{ME})-DNA-Pascal function declared below. Let T' be a set variable that does not occur in P, and let x be the input string.

function $s(T)$
begin
  $T' := T$;
  for $i := 1$ to $p(|x|)$ do $T' := T'/$; (T':=$\backslash T'$ would work as well)
  if $\varepsilon \in T'$ then $s := 1$ else $s := 0$
end

$\square$

The relationships between the powers of different collections of set operations and set tests given by the Lemmas 6, 7, and 10 are presented in the following figure which should be read as the previous ones. Let $\mathcal{O} = \{\mathrm{UN}, \mathrm{EX}, \mathrm{RC}\}$.



The following theorem gives upper bounds for the power of polynomial time computations with the various models including the set operations RC or LC.

Theorem 11 *1.* DNA({UN,EX,IN,EW,RA,LA,CO,RC,LC},{ME,EM})-P $\subseteq \Delta_2^{\mathrm{p}}$,

2. DNA({UN,EX,IN,EW,RA,LA,CO,RC,LC},{ME,EM,SU})-P $\subseteq \Delta_3^{\mathrm{p}}$.

PROOF. [1.]: Following Lemma 6, and Lemma 10, it is sufficient to consider only set operations UN, BX, IN, CO, RC, and LC, and membership tests. Let L be a language decidable by a ({UN,BX,IN,CO,RC,LC},{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 Theorem 4 and Theorem 8 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 cut operations do 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 $a_1a_2\ldots a_m$ by $T_j := \mathrm{In}(m)$ is described by the expression $H = a_1a_2\ldots a_m$. 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

$T_j := T_u \cdot T_v$		$(H_1)(H_2)$		z_1z_2	
$T_j := T_u/$	the expression	$(H_1)/$	is a possible history of the word	$z_1/$	in T_j.
$T_j := \backslash T_u$		$\backslash(H_1)$		$\backslash z_1$	

Define $\omega(H)$ as the word described by the history H and

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

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 $H'\in Q'(H)$. This is achieved 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
$T_j := \mathrm{In}(n)$	for $H' \in Q'(H)$ do if $H' \in \{0,1\}^n$ then $s_j[H'] := 1$
$T_j := T_u \cup T_v$	for $H' \in Q'(H)$ do $s_j[H'] := \max(s_u[H'], s_v[H'])$
$T_j := \mathrm{Bx}(T_u, m, a)$	for $H' \in Q'(H)$ do if $(\omega(H'))(m) = a$ then $s_j[H'] := s_u[H']$ else $s_j[H'] := 0$
$T_j := T_u \cdot T_v$	for $H' \in Q'(H)$ do if $H' = (H_1)(H_2)$ then $s_j[H'] := s_u[H_1] \cdot s_v[H_2]$ else $s_j[H'] := 0$
$T_j := T_u/$	for $H' \in Q'(H)$ do if $H' = (H_1)/$ then $s_j[H'] := s_u[H_1]$ else $s_j[H'] := 0$
$T_j := \backslash T_u$	for $H' \in Q'(H)$ do if $H' = \backslash(H_1)$ 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 a_r $(r = 1, \dots, m)$. Finally, after having simulated 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

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

we obtain $R_P \in \mathrm{NP}$.

The simulation of the DNA-Pascal program P on input x by an ordinary Pascal program with oracle is as in the proofs of Theorem 4 and Theorem 8, 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.

[2.]: The proof of statement 2 is exactly the same 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_2^{p}-oracle and leads to a $\mathrm{P}^{\Pi_2^{\mathrm{p}}}$-computation.

□

After having proven the upper bounds for our models with length-decreasing operations we are ready to give the exact characterization by showing the lower bounds now.

Theorem 12 *1.* DNA({UN,EX,IN,EW,RA,LA,CO,RC,LC},{ME,EM})-P = Δ_2^{p},

2. DNA({UN,EX,IN,RC},{EM})-P = DNA({UN,EX,IN,LC},{EM})-P = Δ_2^{p},

3. DNA({UN,EX,IN,EW,RA,LA,CO,RC,LC},{ME,EM,SU})-P = Δ_3^{p},

4. DNA({UN,EX,IN,RC},{SU})-P = DNA({UN,EX,IN,LC},{SU})-P = Δ_3^{p}.

PROOF. Because of Theorem 11.1 and Lemmas 6, 7 and 10 it is sufficient for the Statements 1 and 2 to prove $\Delta_2^{\mathrm{p}} \subseteq$ DNA({UN,SX,IN,RC},{EM})-P. However, Theorem 9 shows even $\Delta_2^{\mathrm{p}} \subseteq$ DNA({UN,SX,IN},{EM})-P.

Analogously, it remains for the Statements 3 and 4 to prove

$$\Delta_3^{\mathrm{p}} \subseteq \mathrm{DNA}(\{\mathrm{UN},\mathrm{SX},\mathrm{IN},\mathrm{RC}\},\{\mathrm{SU}\})\text{-P}.$$

Define 3-∀SAT as the set of all propositional formulas $H(x_1, \ldots, x_n, y_1, \ldots, y_m)$ in CNF having three variables in each clause and $\forall a_1 \ldots \forall a_n \exists b_1 \ldots \exists b_m \, (H(a_1, \ldots, a_n, b_1, \ldots, b_m) = 1)$. The set 3-∀SAT is complete for Π_2^{p}. It is sufficient to show 3-∀SAT $\in$ DNA({UN,SX,IN,RC},{SU})-P because of

$$\Delta_3^{\mathrm{p}} = \mathrm{P}^{\Pi_2^{\mathrm{p}}} = \mathrm{P}^{\text{3-}\forall\text{SAT}} \overset{!}{\subseteq} \mathrm{P}^{\mathrm{DNA}(\{\mathrm{UN},\mathrm{SX},\mathrm{IN},\mathrm{RC}\},\{\mathrm{SU}\})\text{-P}} = \mathrm{DNA}(\{\mathrm{UN},\mathrm{SX},\mathrm{IN},\mathrm{RC}\},\{\mathrm{SU}\})\text{-P}.$$

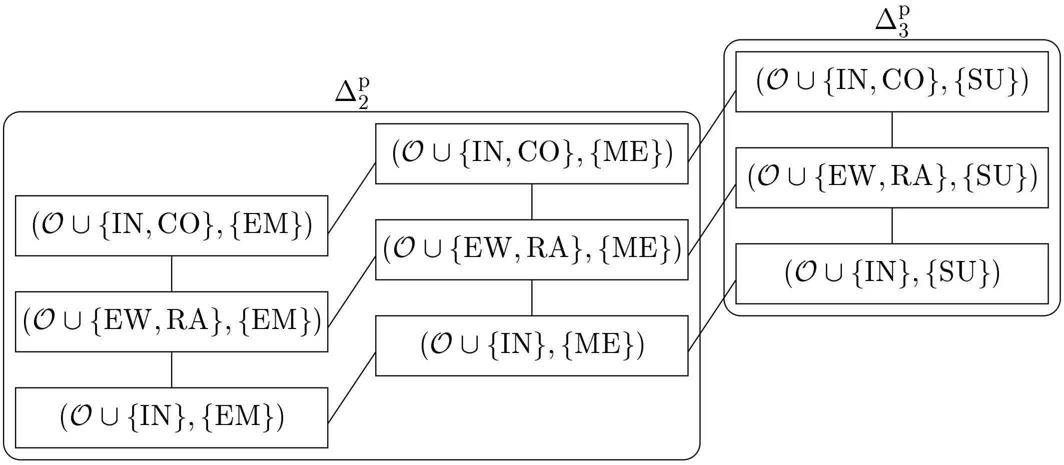
To prove that 3-∀SAT is in DNA({UN,SX,IN,RC},{SU})-P we proceed similar as in the proof of Theorem 9. We construct a ({UN,SX,IN,RC},{SU})-DNA-Pascal program that, given a propositional formula $H(x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m}) \equiv \bigwedge_{i=0}^{k} (x_{\sigma(i,1)}^{a_{i,1}} \vee x_{\sigma(i,2)}^{a_{i,2}} \vee x_{\sigma(i,3)}^{a_{i,3}})$, $\sigma(i,j) \in \{1, \ldots, n+m\}$ and $a_{i,j} \in \{0,1\}$, produces in the set variable T the set of all $w(a_1 \ldots a_n a_{n+1} \ldots a_{n+m})$ such that $H(a_1, \ldots, a_n, a_{n+1}, \ldots, a_{n+m}) = 1$. Then using cut operation RC, it produces the set of all $w(a_1 \ldots a_n)$ (see p. 10 for definition) such that there exists $a_{n+1}, \ldots, a_{n+m}$ with $H(a_1, \ldots, a_n, a_{n+1}, \ldots, a_{n+m}) = 1$. Finally, it is tested whether this is the set of all $w(a_1 \ldots a_n)$ with $a_1, \ldots, a_n \in \{0,1\}$. This is realized by the following program which is a slight modification of the program in the proof of Theorem 9. Note that $\frac{1}{2}(m^2 + 2nm + 11m) = |w(a_1 \ldots a_n a_{n+1} \ldots a_{n+m})| - |w(a_1 \ldots a_n)|$.

begin
  $T := \mathrm{In}(\frac{1}{2}((n+m)^2 + 11(n+m) + 8))$;
  for $j := 1$ to $n+m$ do
    begin
      $T_1 := \mathrm{Sx}(T, 10^{j+1}1\,0\,10^{j+2}1)$;
      $T_2 := \mathrm{Sx}(T, 10^{j+1}1\,1\,10^{j+2}1)$;
      $T := T_1 \cup T_2$
    end
  for $i := 1$ to k do
    begin
      $T_1 := \mathrm{Sx}(T, 10^{\sigma(i,1)+1}1\, a_{i,1})$;
      $T_2 := \mathrm{Sx}(T, 10^{\sigma(i,2)+1}1\, a_{i,2})$;
      $T_3 := \mathrm{Sx}(T, 10^{\sigma(i,3)+1}1\, a_{i,3})$;
      $T := T_1 \cup T_2$;
      $T := T \cup T_3$
    end
  for $j := 1$ to $\frac{1}{2}(m^2 + 2nm + 11m)$ do $T := T/$;
  $T_3 := \mathrm{In}(\frac{1}{2}(n^2 + 11n + 8))$;
  for $j := 1$ to n do
    begin
      $T_1 := \mathrm{Sx}(T_3, 10^{j+1}1\,0\,10^{j+2}1)$;
      $T_2 := \mathrm{Sx}(T_3, 10^{j+1}1\,1\,10^{j+2}1)$;
      $T_3 := T_1 \cup T_2$
    end
  if $T_3 \subseteq T$ then accept
end

Obviously the operation RC can be replaced equivalently by LC if we deal with the reverses $w(a_1 \dots a_{n+m})^R$ rather than with the words $w(a_1 \dots a_{n+m})$.

□

The inclusional relationships given by the Lemmas 6, 7, and 10 and the results of Theorem 12 are represented in the following figure. Again the LA case (LC case, resp.) is left out, as it is analogous to the RA case (RC case, resp.). Let $\mathcal{O} = \{\mathrm{UN}, \mathrm{EX}, \mathrm{RC}\}$.



6 Adding intersection results in PSPACE

Now the question arrises if there are operations or tests that give polynomial-time bounded DNA-Pascal programs the full power of PSPACE. It turns out to be sufficient to add intersection to the set of operations.

Theorem 13

1. DNA($\{$UN,EX,IN,EW,RA,LA,CO,RC,LC,IS$\}$,$\{$ME,EM,SU$\}$)-P = PSPACE.
2. DNA($\{$UN,EX,IN,RC,IS$\}$,$\{$EM$\}$)-P = DNA($\{$UN,EX,IN,LC,IS$\}$,$\{$EM$\}$)-P = PSPACE.

PROOF. In order to proof the upper bound mentioned in the first statement

$$\text{DNA}(\{\text{UN,EX,IN,EW,RA,LA,CO,RC,LC,IS}\},\{\text{ME,EM,SU}\})\text{-P} \subseteq \text{PSPACE}$$

we cite the well known result PARTIME(Pol) = PSPACE. So we have to show that a ($\{$UN,EX,IN,EW,RA,LA,CO,RC,LC,IS$\}$,$\{$ME,EM,SU$\}$)-DNA-Pascal program can be simulated by a parallel polynomially time-bounded Turing machine. Let each string that occurs in a set variable be represented on the tape of one of the components of the parallel Turing machine. Now it is easy to see that the operations and tests in the DNA-Pascal program can be implemented as polynomially time-bounded sequences of that machine.

Next we have to prove the lower bound mentioned in the second statement:

$$\text{PSPACE} \subseteq \text{DNA}(\{\text{UN,EX,IN,RC,IS}\},\{\text{EM}\})\text{-P} = \text{DNA}(\{\text{UN,EX,IN,LC,IS}\},\{\text{EM}\})\text{-P}.$$

To show that we use a slight modification of the famous PSPACE-complete problem QBF. To emphasize that it is an extension of 3-SAT we call it 3-QSAT and define it as the set of all propositional formulas $H(x_1, \ldots, x_n)$ in CNF having three variables in each clause and $\exists x_1 \, \forall x_2 \, \exists x_3 \ldots Qx_n \, (H(x_1, \ldots, x_n) = 1)$, where $Q = \exists$ if n is odd and $\forall$ otherwise. Obviously, the set 3-QSAT is complete for PSPACE, too, so it is sufficient to show 3-QSAT $\in$ DNA($\{$UN,EX,IN,RC,IS$\}$,$\{$EM$\}$)-P, where the chain of arguments is the following

$$\begin{aligned} \text{PSPACE} &= \text{P}^{\text{PSPACE}} \\ &= \text{P}^{\text{3-QSAT}} \overset{!}{\subseteq} \text{P}^{\text{DNA}(\{\text{UN,EX,IN,RC,IS}\},\{\text{EM}\})\text{-P}} = \text{DNA}(\{\text{UN,EX,IN,RC,IS}\},\{\text{EM}\})\text{-P}. \end{aligned}$$

To prove it we construct a ($\{$UN,EX,IN,RC,IS$\}$,$\{$EM$\}$)-DNA-Pascal program that, given a propositional formula $H(x_1, \ldots, x_n) \equiv \bigwedge_{i=0}^{k} (x_{\sigma(i,1)}^{a_{i,1}} \vee x_{\sigma(i,2)}^{a_{i,2}} \vee x_{\sigma(i,3)}^{a_{i,3}}), \sigma(i,j) \in \{1, \ldots, n\}$ and $a_{i,j} \in \{0, 1\}$, produces step by step in the set variable T the sets $U_n, U_{n-1}, \ldots, U_1, U_0$ where

$$U_i =_{\text{df}} \{ \, a_1 a_2 \ldots a_i \mid Q_{i+1} x_{i+1} \ldots Q_n x_n \, (\, H(a_1, a_2, \ldots, a_i, x_{i+1}, \ldots, x_n) = 1 \,) \, \}$$

($Q_j = \exists$ if j is odd and $\forall$ otherwise). Obviously, $H \in$ 3-QSAT if and only if $\varepsilon \in U_0$. Thus, the following program decides $H \in$ 3-QSAT:

```
begin
  T := In(n);                                 (put all possible “assignment strings” in T)
  for i := 1 to k do
    begin
      ⋮                                       (construct U_n in T as in the proof of Theorem 3)
    end;
  for i := n downto 1 do                      (compute U_i)
    if i is even
      then T := Bx(T, i, 0)/ ∩ Bx(T, i, 1)/   (Q_i = ∀)
      else T := T/;                           (Q_i = ∃)
  if T = ∅ then reject else accept
end
```

Notice that the idea of the proof can be realized as well by using LC instead of RC: then one has to deal with the reverses of the assignments.

□

7 A remark on the consumption of resources

The DNA-Pascal programs that we specify in the preceded proofs use exponentially many strings in the set variables. But the question whether this is necessary or whether it might be possible to use only polynomially ‘sized’ set variables is left open. Now we define this kind of restriction.

Definition 14 1. A DNA-Pascal program is called *sparse*, if the set variables used contain only polynomial many strings.

2. Let $\mathcal{O}$ and $\mathcal{T}$ be collections of set operations and set tests, resp. SPARSE-DNA($\mathcal{O}, \mathcal{T}$)-P is the class of problems that can be solved via sparse DNA-Pascal programs which use set operations from $\mathcal{O}$ and tests from $\mathcal{T}$.

Theorem 15

$$\text{SPARSE-DNA}(\{\text{UN,EX,IN,EW,RA,LA,CO,RC,LC,IS}\},\{\text{ME,EM,SU}\})\text{-P} = \text{P}.$$

PROOF. [$\supseteq$]: Obviously correct.

[$\subseteq$]: The information of every set variable T in a sparse DNA-Pascal program can be stored in a single word $w(T)$ of polynomial length. Hence any operation or test on T can be simulated by a constant sequence of instructions on $w(T)$ in polynomial time.

□

8 Restriction to logarithmic space

Often in complexity theory not only time- but also space-bounded models of computation are investigated. In the following section we characterize the power of DNA-Pascal programs that use only logarithmically length-bounded variables.

Definition 16 1. An integer variable is called *logarithmically length-bounded*, if the binary representation of the stored number is logarithmically length-bounded.

2. A set variable is called *logarithmically length-bounded*, if it contains only logarithmically length-bounded strings.

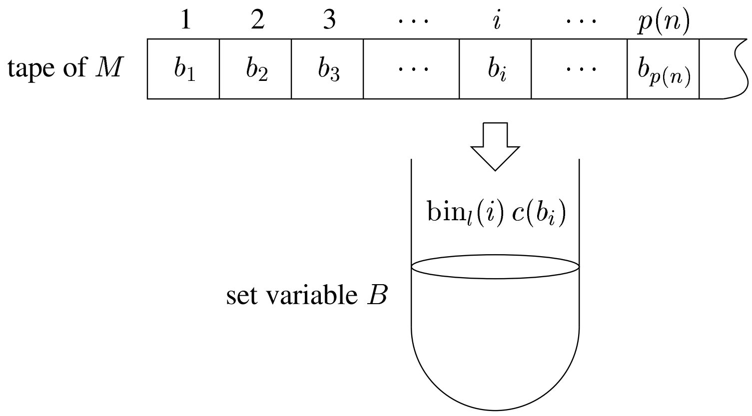
3. Let $\mathcal{O}$ and $\mathcal{T}$ be collections of set operations and set tests, resp. DNA($\mathcal{O}, \mathcal{T}$)-L is the class of problems that can be solved via DNA-Pascal programs which use set operations from $\mathcal{O}$ and tests from $\mathcal{T}$ and only logarithmically length-bounded variables.

Theorem 17 P $\subseteq$ DNA({UN,EX,IN},{EM})-L.

PROOF. As any P-problem can be solved by a deterministic polynomially time-bounded Turing machine it is sufficent for showing the statement, to simulate such a Turing machine by a ({UN,EX,IN},{EM})-DNA-Pascal program using only logarithmically length-bounded variables. Let $M = (Z, A, \delta, z_0, z_1)$ be a polynomially time-bounded Turing machine, where Z is the set of states, A is the alphabet used, $z_0 \in Z$ is the initial state, $z_1 \in Z$ is the final state and $\delta : Z \times \{0,1\} \to Z \times \{0,1\} \times \{-1,0,1\}$ is the transition function (the last component describes the movement of the head). Without loss of generality we assume M to work on one one-sided tape, the input $a_1 \dots a_n$ to be found on the first n tape squares, and if M accepts, the symbol '1' to be written on the first square and '0' otherwise. For a suitable polynomial p, M runs at most $p(n)$ steps.

Now we need some more definitions: let $i, n \in \mathbb{N}$, $x \in \{0,1\}^*$. Then $\mathrm{bin}(i)$ is the binary representation of i; $\mathrm{bin}_n(i)$ the binary representation of i that has exactly length n (if necessary with leading zeros) if $i < 2^n$; x_i is the i-th bit of x. Obviously, for given n, i and j $\mathrm{bin}_n(i)_j$ can easily be computed.

For the simulation of M by a Pascal program, the information of the tape is stored in a set variable B. (Remember that a $p(n)$-time-bounded run of M can not use more than $p(n)$ tape squares.) Let $l =_{\mathrm{df}} |\mathrm{bin}(p(n))|$. As strings in set variables in DNA-Pascal programs are defined over the alphabet $\{0,1\}$, we use a standard encoding $c : A \to \{0,1\}^m$ to transform the symbols from A into a binary block code, where $m := \log|A|$. The squares 1 to $p(n)$ are represented by words in B which have the form $\mathrm{bin}_l(i)\, c(b_i)$ (b_i is the symbol on the i-th square of the tape).



Obviously the length of the words in B is $l + m$. The following fragment initializes B according to the tape of M at the beginning of the run in case of input $a_1 \ldots a_n$; for $i = 1, \ldots, n$ put $b_i = a_i$ and for $i = n+1, \ldots, p(n)$ put $b_i = \square$.

```
begin
  T := In(l + m);                  (T contains all words of length l + m)
  for i := 1 to p(n) do            (for each square i = 1,...,p(n) of the tape
  begin
    T' := T;
    for j := 1 to l do             the words starting with bin_l(i)
      T' := Bx(T', j, bin_l(i)_j); are constructed)
    for j := 1 to m do
      T' := Bx(T', l + j, c(b_i)_j);   (the word bin_l(i) c(b_i) is chosen
    B := B ∪ T'                        and put to B)
    end
end
```

Now B contains exactly $p(n)$ words of the desired form.

To handle the head of M it is usefull to define a counter h that retains the number of the square the head 'looks at'; h is initialized with 1. Additionally we need a variable s to store the actual state, initialized with z_0. The task now is to simulate a transition $z\, b \to z'\, b'\, D$ of M. So if s has the value z we must verify if the word in B starting with $\mathrm{bin}_l(h)$ ends with $c(b)$. In order to do this we 'peel' the h-th word out of B, step by step seperating the words that accord to the positions of $\mathrm{bin}_l(h)$ (collected in T) from those which do not and come into being the rest of B (collected in R). If the h-th word ends with $c(b)$ we have to construct the word $\mathrm{bin}_l(h)\, c(b')$ and then to unify it with the rest of B. Now again B contains $p(n)$ strings that encode the tape of M. The last thing to do is the correction of h: we simply have to add the move $D \in \{-1, 0, 1\}$. This algorithm is realized by the following program.

```
begin
  for z b → z' b' D ∈ δ do                (search for the right instruction of M)
  begin
    if s = z then                          (state is correct)
    begin
      T := B;
      for i := 1 to l do                   ('walk' through the positions of bin_l(h))
      begin                                (seperate the words
        R := R ∪ Bx(T, i, 1 − bin_l(h)_i);   that do not have bin_l(h)_i on the i-th place
        T := Bx(T, i, bin_l(h)_i)            from those which do)
      end
      for i := 1 to m do
      begin
        R := R ∪ Bx(T, l + i, 1 − c(b)_i);   (check whether the h-th tape symbol is not b
        T := Bx(T, l + i, c(b)_i);           or whether it is b)
      end
      B := R;
      if not(T = ∅) then                   (if the symbol is b then
      begin                                perform the transition:)
        s := z';                           (the new state is z')
        T := In(l + m);                    (the h-th word that must end with c(b') now
        for i := 1 to l do
          T := Bx(T, i, bin_l(h)_i);       is constructed here
        for i := 1 to m do
          T := Bx(T, l + i, c(b')_i);
        B := B ∪ T;                        and put to B again)
        h := h + D                         (the head counter is corrected)
      end
    end
  end
end
```

Finally, accepting or rejecting of M is easy to simulate: if M would stop, we just have to check whether the first word in B ends with $c(1)$ or with $c(0)$ (in analogy to checking whether the h-th symbol was b) and then accept or reject.

□

Obviously, nothing harder than polynomially time-bounded problems can be computed within logarithmic space. So it follows from Theorem 17

Corollary 18 DNA({UN,EX,IN},{EM})-L = P.

Notice that if we expand the notion of DNA-Pascal programs to nondeterministic ones, we can simulate nondeterministic polynomially time-bounded Turing machines by nondeterministic ($\{$UN,EX,IN$\}$,$\{$EM$\}$)-DNA-Pascal programs using only logarithmically length-bounded variables. That means this model of parallelism to stand in contrast to those whose nondeterminism is only a special case of parallelism and to those whose nondeterministic versions are not stronger than the deterministic ones.

Definition 19 Let $\mathcal{O}$ and $\mathcal{T}$ be collections of set operations and set tests, resp. DNA$(\mathcal{O}, \mathcal{T})$-NL is the class of problems that can be solved via nondeterministic DNA-Pascal programs which use set operations from $\mathcal{O}$ and tests from $\mathcal{T}$ and only logarithmically length-bounded variables.

Theorem 20 DNA($\{$UN,EX,IN$\}$,$\{$EM$\}$)-NL = NP.

PROOF. The theorem can be shown straightforward following the proof of Theorem 17.

□

Acknowledgement

We are grateful to Heribert Vollmer for bringing Lipton's paper to our attention and to him, Gerhard Buntrock, Ulrich Hertrampf, Katja Jucht, and an anonymous referee for interesting hints.

References

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

[BC94] D. P. 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.