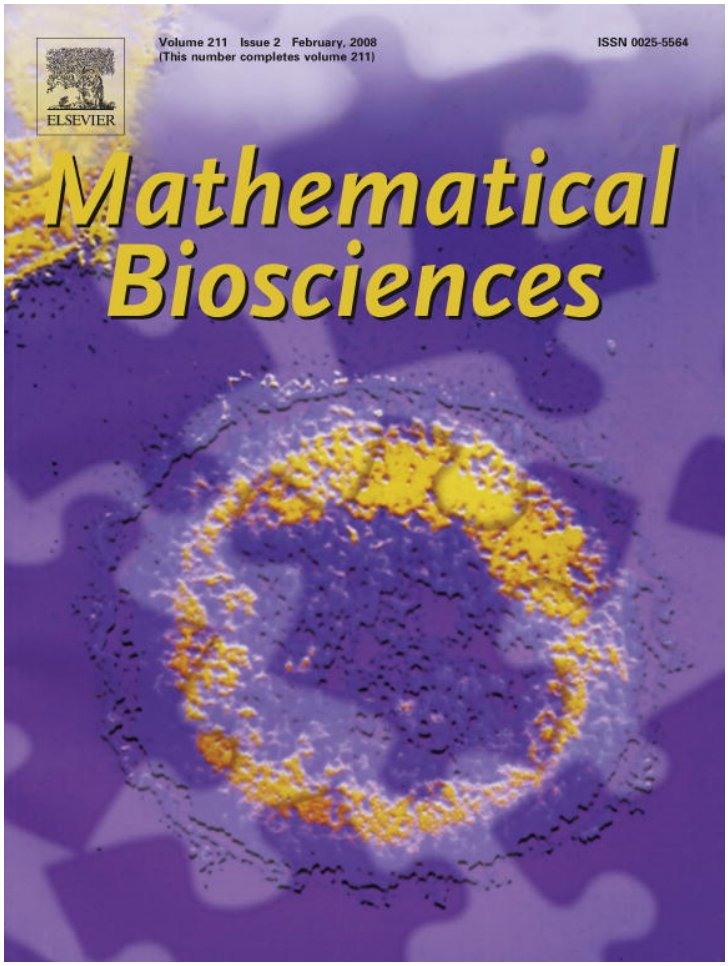

Available online at www.sciencedirect.com
ScienceDirect
Mathematical Biosciences 211 (2008) 282–298
Mathematical Biosciences
www.elsevier.com/locate/mbs

Computing by polymerase chain reaction

Vincenzo Manca *, Giuditta Franco

University of Verona, Department of Computer Science, 15 Strada le Grazie, Verona 37134, Italy

Received 6 June 2006; received in revised form 3 January 2007; accepted 17 August 2007
Available online 6 September 2007

Abstract

A mathematical notation is introduced to represent, at a symbolic level, different mechanisms of DNA recombination, and a 'PCR lemma' is proven by analytically describing the combinatorial properties of the polymerase chain reaction process. This approach led to the discovery of novel techniques, based on a form of PCR which we called cross pairing PCR (briefly XPCR). They were mathematically analyzed and already experimentally proven in different contexts, such as DNA extraction and recombination. Thus, a mathematical analysis of standard methodologies may highlight novel mechanisms of DNA recombination and this can provide new technologies for DNA manipulation.

Keywords: Discrete models of biomolecular processes; DNA algorithms; DNA computing; DNA extraction; DNA recombination; Polymerase chain reaction

1. Introduction

In 1994 Leonard Adleman started the new research field of DNA computing [1]. He showed that an instance of a famous combinatorial problem can be translated in terms of DNA strands, put in a test tube in such a way that, by means of typical laboratory manipulations, a final DNA pool is obtained where the solution of the problem is encoded. Since then, a great deal of research has been carried on and many technical and theoretical achievements have been reached in DNA

* Corresponding author. Tel.: +39 045 802 7981; fax: +39 045 802 7068.
E-mail address: vincenzo.manca@univr.it (V. Manca).

0025-5564/$ - see front matter
doi:10.1016/j.mbs.2007.08.010

computing [12,3–5]. Recently, new research perspectives have emerged that widen the possibilities of this field, among them: DNA self-assembly [21,22,15], DNA automata [2], and tools for DNA and RNA manipulation inspired by algorithmic analyses [17,11].

In the attempt to implement algorithms over a DNA-based 'bio-ware', along to the DNA computing trend, it became increasingly apparent that the logic of DNA operations presents deep combinatorial and algorithmic aspects [8,13]. In particular, polymerase chain reaction, which is a milestone in DNA recombinant technology [19], shows many interesting combinatorial properties. In fact, despite improved methodologies, it remains in some cases a confounding process, and when used in non-standard ways it yields very complex behaviors [20]. Very often, anomalies are ascribed to experimental noise, but, if we frame the process within a rigorous symbolic notation, then non-trivial combinatorial aspects appear and, under suitable hypotheses, a formula can be derived which describes the general form of sequences that are exponentially amplified. This approach is not of merely mathematical interest. On the contrary, it can suggest new methods that enjoy biological relevance for in vitro DNA manipulation. In fact, starting from DNA computing problems [14], we investigated specific methods for DNA extraction and recombination [7,9]. In these attempts, where theoretical issues were supported by the experiments, we realized that a special kind of PCR, called *Cross Pairing PCR* or shortly XPCR, can be the basis for new algorithms that solve a wide class of DNA extraction and recombination problems. Within a basic DNA symbolic notation, we will show a crucial computational property of polymerase chain reaction, referred to as PCR Lemma, which suggested us the idea of Cross Pairing PCR.

2. DNA notation

Firstly, we recall and extend some standard notation from Formal Language Theory (FLT) [18] in order to formalize fundamental operations related to the structure of DNA molecules (see Table 1).

Let us consider the usual alphabet of bases $\mathcal{B} = \{A, T, C, G\}$. The set $\mathcal{B}^*$ of *strings* over this alphabet is comprised of the sequences (words) that can be arranged with these four symbols (letters). Strings of $\mathcal{B}^*$ will be indicated by Greek letters. On these strings, a binary associative operation of *concatenation* is defined, that given two strings of $\mathcal{B}^*$ yields a new string where all the symbols of the second sequence are put, in the given order, after the last symbol of the first one. Concatenation between two strings α and β is denoted by the juxtaposition $\alpha\beta$. The length of a string α is the number of its symbol occurrences (each symbol is counted as many times as it occurs) and is indicated by $|\alpha|$. A special string is that of length 0, denoted by λ, that is the *empty* string, where no symbol occurs (an abstract notion similar to zero for numbers). Symbols are special strings of length 1. Mathematically speaking, the structure $\mathcal{B}^*$ is referred to as the *free monoid* over the alphabet $\mathcal{B}$. Any subset of $\mathcal{B}^*$ is a (formal) *language* over the alphabet $\mathcal{B}$. Since languages are sets, all the usual set theoretical notions extend to languages (such as membership $\in$, inclusion $\subseteq$, empty set $\emptyset$).

The symbol of α that occurs in position i ($1 \leqslant i \leqslant |\alpha|$) is denoted by $\alpha(i)$, the sequence of symbols of α occurring (in the given order) from position i to position j ($1 \leqslant i \leqslant j \leqslant |\alpha|$) is denoted by $\alpha[i,j]$ and is called a *substring* of α. In particular, it is called a *prefix* if $i = 1$, and a *suffix* if $j = |\alpha|$. A string with prefix α and suffix β is also denoted by $\alpha \ldots \beta$. The *complementation* function, denoted

Table 1
Basic DNA operations

$\alpha[i,j]$	Substring
$\alpha\beta$	Concatenation
$(\alpha)^c$	Complementation
$rev(\alpha)$	Reversing
$mir(\alpha)$	Mirroring
$\alpha \bowtie \beta$	Overlapping
$\alpha \rfloor \beta$	Overlap Concatenation
$\alpha][\beta$	Hybridization
$\frac{\alpha}{\beta}$	Pairing
$\langle \alpha \rangle$	Blunt Pairing

by the superscript c, is defined on $\mathcal{B}$ by $A^c = T$, $T^c = A$, $C^c = G$, $G^c = C$. It extends naturally to $\mathcal{B}^*$ by the conditions $\lambda^c = \lambda$ and $(\alpha\beta)^c = \alpha^c\beta^c$. Another operation on $\mathcal{B}^*$ is the *reverse* operation *rev* such that, for every $X \in \mathcal{B}$, $rev(X) = X$, $rev(\alpha X) = Xrev(\alpha)$, and $rev(\lambda) = \lambda$. Of course, reversing and complementation are *involutive* and *commute*, that is, $rev(rev(\alpha)) = \alpha$, $(\alpha^c)^c = \alpha$ and $rev(\alpha^c) = (rev(\alpha))^c$. The *mirroring* of α is defined by $mir(\alpha) = rev(\alpha^c)$ and it is an involutive operation; we abbreviate $mir(\alpha)$ as $\bar{\alpha}$. In DNA molecules, an intrinsic concatenation verse is given which goes from the Phosphate to the Oxydryl (that, respectively, correspond to Carbon 5′ and 3′ positions in the sugar DNA backbone). This verse is symbolically denoted by an arrow at one side of non-null strings α, like in $\alpha\rightarrow$, that can be seen as an indication of the extremity where Oxydryl terminal is located (while Phosphate being at the other extremity). However, when we omit the arrow, it is intended that it is on the right end of the string (i.e., usual reading from the left to the right corresponds to the 5′–3′ verse).

An *exact pairing* (symmetric) relation $\|$ is defined over $\mathcal{B}^*$ such that $\alpha\|\beta$ if $\beta = mir(\alpha)$. Let γ be the longest string such that α and β include γ and $\bar{\gamma}$, respectively. Let h_{hybr} be a non-null length that we call *hybridization threshold* (it represents a biological parameter we leave unspecified). If $|\gamma| > h_{\mathrm{hybr}}$, then we say that α and β *pair by hybridizing* on γ (or shortly, that they *hybridize*) and write $\alpha]\gamma[\beta$. In this case a double DNA string composed of α and β is defined, that will be denoted by $\frac{\alpha}{rev(\beta)}$. In this fraction notation the following concatenation verses are implicitly assumed $\frac{\alpha\rightarrow}{\leftarrow rev(\beta)}$, that is, the Oxydryl terminal of the inferior string is located on the opposite side with respect to that of the superior string. Sometimes fraction notation is a way to make explicit the verse of single strings. In fact, we put $\frac{\alpha}{\lambda} = \alpha \rightarrow$ (superior notation), $\frac{\lambda}{\alpha} = \leftarrow rev(\alpha)$ (inferior notation), and for notation completeness it is assumed that $\frac{\lambda}{\lambda} = \lambda$.

We write $\alpha][\beta$ if α and β hybridize and $\alpha]\not[\beta$ when they do not hybridize. Pairing is a crucial operation of double strings formation, where the phenomena of *bilinearity*, *complementarity* and *antiparallelism* are jointly involved [8]. It is a partial operation because it is defined on two strings only when they hybridize. The set of single or double DNA strings will be denoted by $\mathcal{B}^*/\mathcal{B}^*$. By the symmetry of the pairing relation, double strings satisfy the following equation: $\frac{\alpha}{rev(\beta)} = \frac{\beta}{rev(\alpha)}$. This is an important aspect of double strings, that formalizes the mobility of DNA strands in a liquid environment. On the other hand, any α hybridizes exactly with $\bar{\alpha}$ by definition of exact pairing, hence the double string $\frac{\alpha}{rev(\bar{\alpha})}$ is defined, which is called a *blunt* double string and is denoted by $\langle\alpha\rangle$. We note that

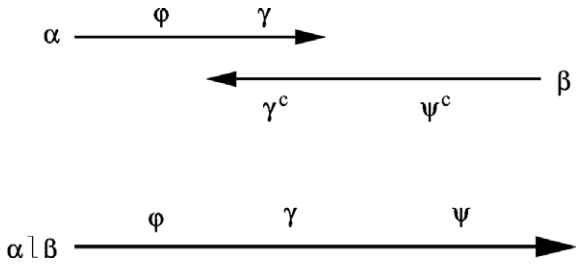

Fig. 1. Two strings which overlap and their overlap concatenation.

$$\langle\alpha\rangle = \frac{\alpha}{rev(\bar{\alpha})} = \frac{\alpha}{rev(rev(\alpha^c))} = \frac{\alpha}{\alpha^c}$$

and as a consequence of the equations above we obtain $\langle\bar{\alpha}\rangle = \langle\alpha\rangle$, in fact we have that

$$\langle\bar{\alpha}\rangle = \frac{\bar{\alpha}}{rev(\bar{\bar{\alpha}})} = \frac{\bar{\alpha}}{rev(\alpha)} = \frac{\alpha}{rev(\bar{\alpha})} = \frac{\alpha}{rev(rev(\alpha^c))} = \langle\alpha\rangle.$$

The *overlapping* of α, β, indicated by $\alpha \bowtie \beta$ is defined as the longest substring γ such that $\alpha = \varphi\gamma$ and $\beta = \overline{\gamma}\psi$ for some strings φ, ψ. If $\alpha \bowtie \beta \neq \lambda$ we say that α and β *overlap*. Let φ and ψ be the strings such that $\alpha = \varphi\gamma$, $\gamma = \alpha \bowtie \beta$, and $\bar{\beta} = \gamma\psi$, then $\alpha\rfloor\beta$ is the string $\varphi\gamma\psi$ which is called the *overlap concatenation* of α and β. Fig. 1 shows two strings which overlap and their overlap concatenation. Of course if α and β overlap, then $|\varphi\gamma\psi| < |\alpha\beta|$.

We call *strand* any object s on which an operation *type* is defined which assigns to it a string of $\mathcal{B}^*/\mathcal{B}^*$. In particular, if *type*(s) is a single string, then we say that s is a *single strand*, while if *type*(s) is a double string, then we say that s is a *double strand*. Intuitively, strands are DNA molecules, which as physical objects are different from the base sequence they realize. In our terminology, strands having the same base sequence are strands with the same type, as they are associated by type operation to one formal string. We restrict ourselves to consider only single or double strands. In fact, for the needs of the following discussion, we may avoid to consider more complex forms of DNA strands which combine more than two strands in a single DNA molecule.

We consider a DNA pool P as a set of strands, or equivalently, as a *multiset* of single or double strings of $\mathcal{B}^*/\mathcal{B}^*$, which is specified by a *multiplicity* function $mult_P$ from $\mathcal{B}^*/\mathcal{B}^*$ to natural numbers. In fact, $mult_P(\eta) = n$ means that the pool P contains n (indiscernible) strands of type η. We write $P = \{n_1{:}\eta_1, n_2{:}\eta_2, \ldots, n_k{:}\eta_k\}$ when $mult_P(\eta_1) = n_1$, $mult_P(\eta_2) = n_2, \ldots, mult_P(\eta_k) = n_k$ and $mult_P(\eta) = 0$ for $\eta \notin \{\eta_1, \eta_2, \ldots, \eta_k\}$. Being DNA pools multisets, standard multiset operations of sum and difference, denoted by $+$, $-$, correspond to the sum and difference of their multiplicity functions.

In virtue of these two ways of considering a pool, we can use (ambiguously) both notations: $\eta \in P$, for a string $\eta \in \mathcal{B}^*/\mathcal{B}^*$, meaning $mult_P(\eta) \neq 0$, and $s \in P$, when s is a strand of P. The type of a pool P is the set of strings (a language):

$$Type(P) = \{\eta \in \mathcal{B}^*/\mathcal{B}^* \,|\, \eta \in P\} = \{type(s) \in \mathcal{B}^*/\mathcal{B}^* \,|\, s \in P\}.$$

We remark the difference between *type*, which assigns to a strand a string, and *Type* (with capital T) which assigns, to a pool of DNA strands, the set of types of its strands. Although strings and strands are different things, very often the two terms are used almost synonymously. In fact, any

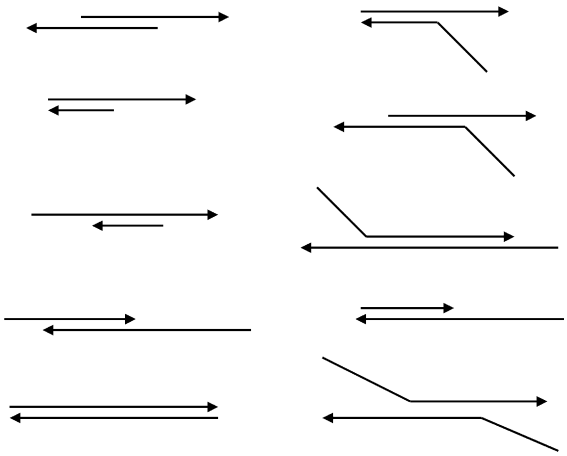

Fig. 2. From the top, according to the usual sticky end notation: Left: 3′3′, 3′, 3′5′, 5′5′, Blunt; Right: Y, 3′Y, 5′Y, 5′, YY.

string is physically implemented by strands having its type, and conversely, the expression 'a string', in a given context, could refer to a physical occurrence of a string, which is just a strand.

Fig. 2 shows 10 different forms of double DNA strings, where parallelism between lines refers to hybridization between strings. Y forms are double strands where corresponding extremal parts of single strings do not hybridize (in YY forms this happens on both sides). Of course, these forms do not cover all possible DNA forms, but they identify all the possibilities of pairing two different strands (circular forms, heteroduples and hairpins are not considered).

The DNA pool operations defined in Table 2 are to be considered as high level mechanisms, analogous to the basic operations of a high level programming language. This implies that the algorithms we specify by means of them are not exactly experimental protocols, but rather computational procedures, implementable by laboratory procedures. These DNA algorithms are naturally expressed in terms of a *Test Tube register Language* (shortly TTL), where registers are Test Tubes which contain DNA pools, rather than numbers, and DNA pool operations apply to them. The symbol := denotes the usual *assignment* command typical of imperative languages. In an

Table 2
Basic requirements on DNA pool operations

$mix(P_1, P_2) = P_1 + P_2$
$split(P) = (P_1, P_2) \Longleftrightarrow P = P_1 + P_2,\ Type(P_1) = Type(P_2) = Type(P)$
$length(P) = \{\|\eta\| \,\|\langle\eta\rangle \in Type(P)\}$
$separate(P, n) = \{s \in P \,\|\, type(s) = \eta,\ \|\eta\| = n\}$
$Type(denature(P)) \supseteq \{\alpha \,\|\, \frac{\alpha}{\beta} \in Type(P)\}$
$Type(hybridize(P)) \supseteq \{\frac{\alpha}{rev(\beta)} \,\|\, \alpha, \beta \in Type(P), \alpha][\beta\}$
$Type(extend(P)) \supseteq \{\frac{\alpha\gamma\beta}{(\delta\gamma\beta)^c} \mid \frac{\alpha\gamma}{(\delta\gamma\beta)^c} \in P\}$
$Type(infix(P, \gamma, \delta))\| \subseteq \{\langle\gamma\alpha\ \delta\rangle \,\|\, \langle\alpha\rangle \in P\}$

assignment $P := op(Q)$, the operation op is intended to be applied to the content of the test tube Q and the resulting DNA pool becomes the content of the test tube P.

The exact 'microscopic' effect of operations in Table 2 is not defined, because in many cases it is hard to say exactly what are the multisets before and/or after the application of these operations. However, despite such a kind of incomplete knowledge at a microscopic level, DNA algorithms can be designed which are based on these operations. In fact, it is enough to assume that, after performing a given DNA operation, an input pool P is transformed into a pool P' where a specific relationship holds between the types of P and P' (see Table 2).

3. Cross pairing PCR

PCR is one of the most important and efficient tool in biotechnological manipulation and analysis of DNA molecules, where the polymerase enzyme implements a very simple and efficient duplication algorithm on double oriented strings. The bilinearity of DNA molecules and the antiparallel orientation of their two linear components are essential aspects of the logic underlying the polymerase chain reaction process [8,13]. The computational schema of PCR in TTL notation is given in Table 3, while its combinatorial schema in terms of string transformations is given in Table 4.

We denote by $PCR(P,n)$ the DNA pool obtained by applying to the pool P the procedure of Table 3 (that express the usual PCR process). The parameter n denotes the number of cycles of the fundamental PCR step. We avoid to mention it when its value is not essential in the discussion. We write also $PCR(P,\gamma,\delta)$ as an abbreviation of $PCR(P \cup \{p{:}\gamma, q{:}\delta\})$, for some numbers p, q. It is well known that if $P = \{m : \langle \gamma\varphi\bar{\delta}\rangle, p : \gamma, p : \delta\}$, where $\langle \gamma\varphi\bar{\delta}\rangle$ is called the *target molecule* of PCR

Table 3
PCR algorithm

$PCR(P,n)=$
let $Type(P) = \{\langle \gamma \ldots \delta\rangle, \gamma, \bar{\delta}\}, n$ integer;
input P;
for $i = 1,n$ **do**
begin
  $P := denature(P)$;
  $P := hybridize(P)$;
  $P := extend(P)$;
end;
output P.

Table 4
Combinatorial schema of polymerase chain reaction

$\frac{\gamma\zeta\delta}{\gamma^c\zeta^c\delta^c} \longrightarrow \frac{\gamma\zeta\delta}{\lambda}, \frac{\lambda}{\gamma^c\zeta^c\delta^c}$	*template denaturation*
$\frac{\gamma\zeta\delta}{\lambda}, \frac{\lambda}{\gamma^c\zeta^c\delta^c}, \frac{\gamma}{\lambda}, \frac{\lambda}{\delta^c} \longrightarrow \frac{\gamma}{\gamma^c\zeta^c\delta^c}, \frac{\gamma\zeta\delta}{\delta^c}$	*primer hybridization*
$\frac{\gamma}{\gamma^c\zeta^c\delta^c}, \frac{\gamma\zeta\delta}{\delta^c} \longrightarrow \frac{\gamma\zeta\delta}{\gamma^c\zeta^c\delta^c}, \frac{\gamma\zeta\delta}{\gamma^c\zeta^c\delta^c}$	*polymerase extension*

and γ, δ are called *primers*, then $PCR(P,n) \approx \{p+m : \langle \gamma\varphi\bar{\delta}\rangle\}$ where $n \approx \log_2(p)$ (the symbol $\approx$ means equal with some approximation), and only a minor quantity of strands in P have types different from $\langle \gamma\varphi\bar{\delta}\rangle$. In general, when at end of a PCR process, the amount of strands of type η is increased according to a ratio which is a power of the number n of the performed steps, then we say that η has been *exponentially amplified* by that PCR.

The polymerase extension *ext* of a single string $\alpha\gamma$ according to a template η is *properly* defined as $ext(\alpha\gamma,\eta) = \alpha\gamma\beta$ when $\eta = \bar{\beta}\bar{\gamma}\bar{\delta}$ and $\alpha\gamma]\gamma[\bar{\beta}\bar{\gamma}\bar{\delta}$ (by assuming that γ does not occur as a substring of α, β, and δ), that is

$$ext\left(\alpha\gamma, \frac{\lambda}{\delta^c\gamma^c\beta^c}\right) = \alpha\gamma\beta$$

otherwise, the extension is not proper and we set

$$ext(\alpha\gamma,\eta) = ext\left(\alpha\gamma, \frac{\lambda}{rev(\eta)}\right) = \alpha\gamma.$$

When $\alpha]\![\beta$ it is helpful to use, in combination with the fraction notation, a more concise notation for *ext*, by writing $\frac{ext(\alpha)}{rev(\beta)}$ instead of $\frac{ext(\alpha,\beta)}{rev(\beta)}$ and $\frac{\alpha}{rev(ext(\beta))}$ instead of $\frac{\alpha}{rev(ext(\beta,\alpha))}$. It follows directly from antiparallelism of double strings that, when $\alpha]\![\beta$, then $ext(\alpha,\beta) = ext(\alpha, ext(\beta,\alpha))$; this consistently allows us to set

$$\frac{ext(\alpha)}{rev(ext(\beta))} = \frac{ext(\alpha,\beta)}{rev(ext(\beta,\alpha))}.$$

With this notation the pool *extend*(P), previously defined (see Table 2), turns to be:

$$\left\{\frac{ext(\alpha)}{rev(ext(\beta))}\middle|\frac{\alpha}{rev(\beta)} \in P\right\}.$$

The main feature of classical PCR can be expressed in terms of *ext* by the following schema which explains the exponential power of this process:

$$\langle \gamma\cdots\delta\rangle \Rightarrow \frac{ext(\gamma)}{(\gamma\cdots\delta)^c}, \quad \frac{\gamma\cdots\delta}{rev(ext(\bar{\delta}))} \Rightarrow \langle \gamma\cdots\delta\rangle, \langle \gamma\cdots\delta\rangle.$$

PCR processes are easily representable by diagrams like those in Figs. 3 and 4, where dotted arrows represent the extend operation performed by polymerase enzyme.

All the possible PCR diagrams which result from the different forms of target strings and from the different positions where primers may hybridize, are close to 100. Therefore a natural question arises: in which cases does PCR provide exponential amplification? And, in these cases, is it possible to characterize, in general terms, the form of strings that are exponentially amplified? Our DNA notation allows us to answer these questions with the following lemma, which exactly describes the form of those blunt strings which are exponentially amplified by a PCR process.

PCR Lemma. *Let P be a DNA pool such that $Type(P) = \left\{\frac{\alpha}{rev(\beta)}\right\}$ and let γ and δ be two strings such that $\gamma]\![\bar{\delta}$. Then $PCR(P,\gamma,\bar{\delta})$ exponentially amplifies if and only if one of the following cases holds*:

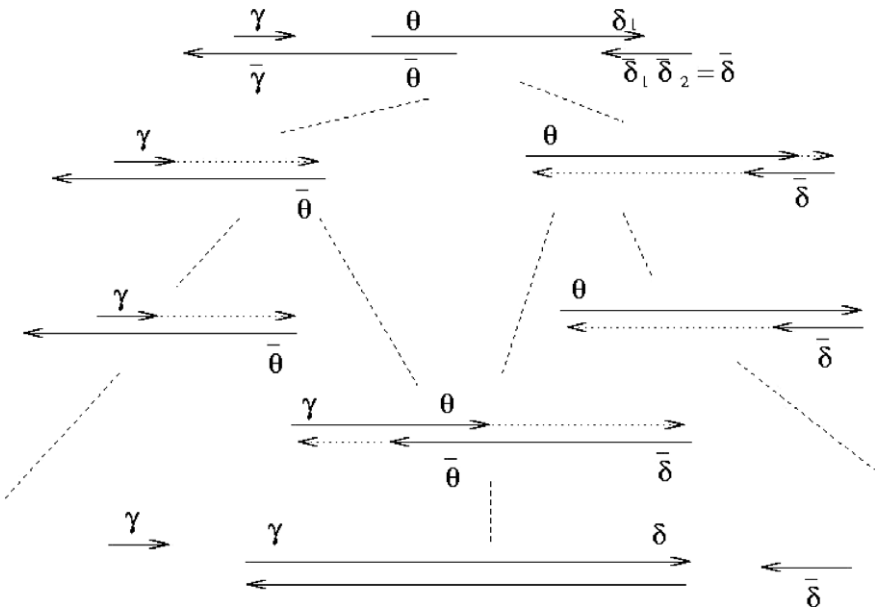

Fig. 3. PCR on a 3′3′ form along with one internal and one external primer. Reading from the top: after the hybridization of the primers (short arrows), and the formation of their extensions, these extensions overlap and, with the further extensions of the overlapping strings, a blunt string is formed where primers corresponds to the two (up and down) extremities.

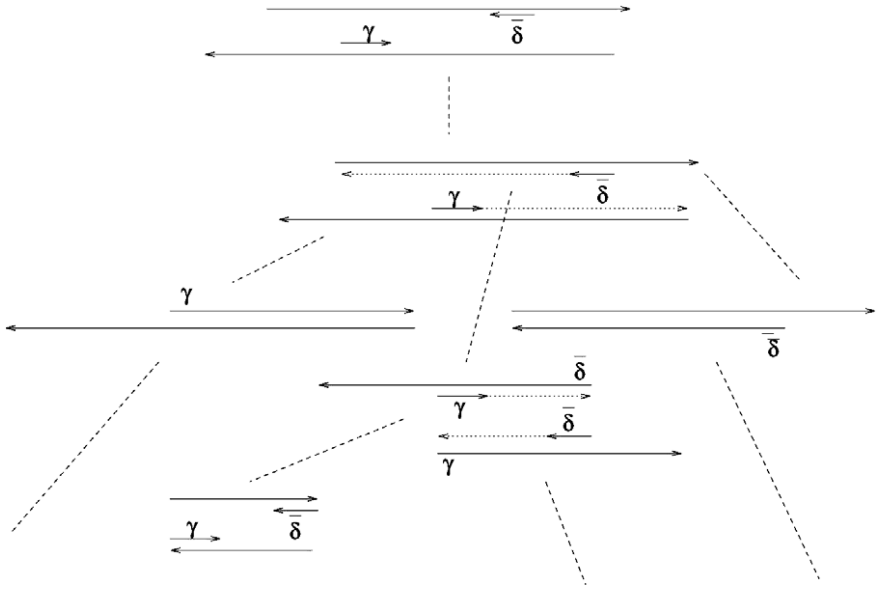

Fig. 4. Standard PCR with two internal primers. From the top. (First level) The target double string and the primers put in correspondence of the corresponding portion they can hybridize with. (Second level) Hybridization of the primers. (Third level) Extensions of the primers with respect to the corresponding templates. (Fourth level) On left and right primers hybridize with the templates of the previous level, while on the center primers hybridize with the extensions formed at the previous level. (Fifth level) The extensions of primers hybridized at the previous level which came from the center forms are blunt form with a primer as a prefix (up string) and the other primer as a suffix (down string).

Case I $ext(\gamma,\beta) \bowtie ext(\bar{\delta},\alpha) \neq \lambda$.
The PCR$(P,\gamma,\bar{\delta})$ *provides an exponential amplification of the following type*
$\langle ext(\gamma,\beta)\rceil\lfloor ext(\bar{\delta},\alpha)\rangle$.

Case II $\alpha \bowtie \beta \neq \lambda, \gamma \bowtie ext(\beta,\alpha) \neq \lambda, \bar{\delta} \bowtie ext(\alpha,\beta) \neq \lambda$.
The PCR$(P,\gamma,\bar{\delta})$ *provides an exponential amplification of the following type*
$\langle ext(\gamma, ext(\beta,\alpha))\rceil\lfloor\langle ext(\bar{\delta}, ext(\alpha,\beta))\rangle\rangle$.

Case III $\bar{\delta} \bowtie ext(\gamma,\beta) \neq \lambda$.
The PCR$(P,\gamma,\bar{\delta})$ *provides an exponential amplification of the following type*
$\langle\bar{\delta}\rceil\lfloor ext(\gamma,\beta)\rangle$.

Case IV $\gamma \bowtie ext(\bar{\delta},\alpha) \neq \lambda$.
The PCR$(P,\gamma,\bar{\delta})$ *provides an exponential amplification of the following type*
$\langle\gamma\rceil\lfloor ext(\bar{\delta},\alpha)\rangle$.

Proof. *If*: In all the cases we are in the situation of overlapping illustrated by Fig. 1, and it follows immediately that a further extension of each string with respect to its paired string provides a blunt string having γ as prefix and δ as suffix. Thus an exponential amplification of this blunt string is a direct consequence of the basic PCR mechanism reported in Table 4. Four cases corresponding to the cases of this lemma are reported in Fig. 7.

Only if: First at all, we notice that, in order to get an exponential amplification by PCR, the process has to necessarily generate a strand of a type which is a proper extension of a primer and, at same time, hybridizes with the other primer. Now we show that if $PCR(P,\gamma,\bar{\delta})$ provides an exponential amplification of some type, then we are necessarily in one of the cases given in the lemma. In fact, we consider the following possibilities:

(i) $ext(\gamma,\beta) \neq \gamma$ and $ext(\bar{\delta},\alpha) = \bar{\delta}$
(ii) $ext(\gamma,\beta) = \gamma$ and $ext(\bar{\delta},\alpha) \neq \bar{\delta}$
(iii) $ext(\gamma,\beta) \neq \gamma$ and $ext(\bar{\delta},\alpha) \neq \bar{\delta}$
(iv) $ext(\gamma,\beta) = \gamma$ and $ext(\bar{\delta},\alpha) = \bar{\delta}$

If the possibility (i) holds, we can distinguish two alternative situations: (ia) the primer $\bar{\delta}$ hybridizes with $ext(\gamma,\beta)$ or (ib) no such a hybridization happens. If (ia) holds, then we are in the Case III. If (ib) holds, since primer $\bar{\delta}$ does not extend with respect to the template α, then the only chance for an extension of $\bar{\delta}$ could be an hybridization of it with an extension of $ext(\gamma,\beta)$, but the only possible extension of $ext(\gamma,\beta)$ is $ext(ext(\gamma,\beta),\beta) = ext(\gamma,\beta)$, and according to (ib) $\bar{\delta}$ does not hybridize with $ext(\gamma,\beta)$. Therefore in the case (ib) no exponential amplification can occur.

The situation of (ii) can be analyzed in a way analogous to (i) which provides the case IV.

If the possibility (iii) holds, both extensions of primers are proper. Let us consider the extension $ext(\gamma,\beta)$ and the primer $\bar{\delta}$, these two stings may hybridize or not. If they hybridize we are in the Case I of the lemma. If they do not hybridize, then consider $ext(\bar{\delta},\alpha)$ and the primer γ. Again, if they hybridize we are in the Case I of the lemma, but if they do not hybridize no primer extension can hybridize with the other primer, therefore no exponential amplification can occur.

In the last possibility (iv), if some of the hypotheses of Case II is not valid, then no extension process can occur, but if all these hypotheses occur, then we fall in this case. □

The phenomenon of the blunt form of PCR exponential amplification is empirically well known, but it is interesting that it is a mathematical consequence of the combinatorial mechanism on which PCR is based. The following corollary is a direct consequence of PCR lemma.

Corollary. *If there is any exponential amplification, then at most at third level of the PCR process a blunt string appears which is a seed of exponential amplification.*

The content of the previous lemma and corollary may be informally paraphrased in this way: the first or the second primer extensions overlap each other or with the primers if and only if, at most at the third PCR step, a blunt strand is produced which is a seed of an exponential amplification. For example, if we consider PCR diagrams such as those in Fig. 4, where each level represents both the hybridization and extension of strands and branching represents denaturation, from the equation $\langle \varphi \rceil \lfloor \psi \rangle = \frac{ext(\varphi)}{rev(ext(\psi))}$ one may deduce that, at most at third level in the diagram, a blunt string appears which is a seed of exponential amplification.

An interesting example where PCR lemma does not apply is displayed in Fig. 5 which is the case of a YY form such that both the primers γ and $\bar{\delta}$ hybridize with it, but no amplification is provided by $PCR(P, \gamma, \bar{\delta})$.

The analysis developed by PCR Lemma points out the importance of overlap concatenation in the string recombination producing PCR amplification.

The importance of the overlap concatenation in some string recombination producing PCR amplification suggested us an interesting form of PCR where, rather than just one amplification target string, we put two target DNA double strings of types $\langle \alpha\varphi\gamma \rangle$ and $\langle \gamma\psi\beta \rangle$ in a test tube.

Let us analyze what happens if, starting from such a pool, a PCR with primers α, $\bar{\beta}$, is performed. In this case, after denaturation the following strings will be present in the pool:

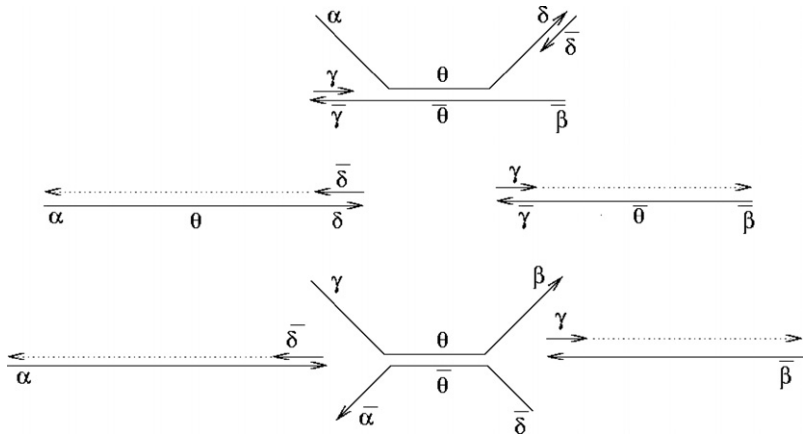

Fig. 5. A YY form where PCR lemma does not apply.

$$\frac{\alpha\varphi\gamma}{\lambda}, \quad \frac{\lambda}{(\alpha\varphi\gamma)^c}, \quad \frac{\gamma\psi\beta}{\lambda}, \quad \frac{\lambda}{(\gamma\psi\beta)^c}, \quad \alpha, \quad \bar{\beta}.$$

If the temperatures at which α, β and γ hybridize with their mirror strings are 'close enough', then the following hybridizations are obtained:

$$\frac{\alpha\varphi\gamma}{(\gamma\psi\beta)^c}, \quad \frac{\alpha}{(\alpha\varphi\gamma)^c}, \quad \frac{\gamma\psi\beta}{\beta^c}.$$

At this point polymerase can directly extend strings in such a way that the blunt string $\langle\alpha\varphi\gamma\psi\beta\rangle$ is produced, which results from the *overlap concatenation* of $\alpha\varphi\gamma$ and $\overline{\gamma\psi\beta}$ (that is $\alpha\varphi\gamma\rceil\lfloor\overline{\gamma\psi\beta} = \alpha\varphi\gamma\psi\beta$) with two more strings equal to the initial target strings. We call this special kind of *PCR cross pairing PCR* or shortly *XPCR*; its combinatorial schema is displayed in Table 5. It is a technique implementing an operation which theoretically can be seen as a special kind of Tom Head's *null context splicing rule* given in his seminal paper [10], where a Formal Language Theory perspective of DNA strings was introduced (see also [16]).

In conclusion,

if $Type(P) = \{\langle\alpha\varphi\gamma\rangle, \langle\gamma\psi\beta\rangle\}$, then

$$Type(PCR(P, \alpha, \beta)) \approx \{\langle\alpha\varphi\gamma\psi\beta\rangle\}$$

(types of strands different from $\langle\alpha\varphi\gamma\psi\beta\rangle$ are present in a minor quantity). From an experimental point of view, we observe that, if a PCR of primers α, β is applied to a pool containing both strings $\xi_1 = \langle\alpha\delta\gamma\rangle$ and $\langle\gamma\theta\beta\rangle$, then $\zeta = \langle\alpha\delta\gamma\theta\beta\rangle$ is a product of this PCR (see the combinatorial schema of XPCR in Table 5). We tested *XPCR* in several different experimental conditions and all the times it provided correct results [7,9]. In most cases, the amplification signal is very clear and a small noise consisting of unspecific products is reported in the electrophoresis results. Surprisingly enough, on the basis of the simple mechanism of *XPCR*, we have been able to build more sophisticated procedures, which find two main kinds of applications in DNA Extraction and DNA Recombination.

Table 5
Combinatorial schema of cross pairing polymerase chain reaction

$\langle\alpha\varphi\gamma\rangle, \langle\gamma\psi\beta\rangle, \alpha, \bar{\beta}$
$\Downarrow$
$\frac{\alpha\varphi\gamma}{\lambda}, \frac{\lambda}{(\alpha\varphi\gamma)^c}, \frac{\gamma\psi\beta}{\lambda}, \frac{\lambda}{(\gamma\psi\beta)^c}, \alpha, \bar{\beta}$
$\Downarrow$
$\frac{\alpha\varphi\gamma}{(\gamma\psi\beta)^c}, \frac{\gamma\psi\beta}{(\alpha\varphi\gamma)^c}, \alpha, \bar{\beta}$
$\Downarrow$
$\frac{\alpha\varphi\gamma}{(\gamma\psi\beta)^c}, \frac{\alpha}{(\alpha\varphi\gamma)^c}, \frac{\gamma\psi\beta}{\beta^c}$
$\Downarrow$
$\langle\alpha\varphi\gamma\psi\beta\rangle, \langle\alpha\varphi\gamma\psi\beta\rangle$

4. DNA extraction by XPCR

DNA extraction is a fundamental procedure where the 'good solutions' are discriminated in a space of possible solutions. For example, given a DNA pool consisting of a family of genes with an indefinite identity (their sequences are not known), one might be interested in extracting the subfamily of those genes where a given subsequence γ occurs, that for instance enjoys an important biological meaning. The classic extraction procedure *by affinity* uses a probe $\bar{\gamma}$ which is 'marked' in such a way that, after denaturation, single strands where γ occurs hybridize with the probe, and so are selected from the original pool. Here we show a different way of performing extraction, based on *XPCR* [7], as it is outlined in Table 6.

The main idea of the algorithm is as follows. Consider all the lengths of strands in a given pool. For each length n, pieces of strands which include the substring γ in their types are copied. This is performed by means of usual PCR to amplify both strands of type $\langle\alpha\gamma\rangle$ and $\langle\gamma\beta\rangle$. Strings shorter than n are then separated by length and finally joined by an overlapping concatenation performed by XPCR. In the joining process, pieces that do not reach a suitable length are removed. For this reason, the process must be iterated for each length of strings of the initial pool. The algorithm reported in Table 6 provides all the strings where γ occurs, elongated by the prefix α and by the suffix β. This algorithm was tested in vitro where, from a very heterogeneous pool, all and only the types of strands having substrands of a given type were extracted, that is, extraction resulted to be correct and complete [7].

A worthwhile warning about the XPCR-Extract algorithm is given by the following observation. If in a family of initial genes there are two different genes, say $\langle\varphi\gamma\psi\rangle$ and $\langle\sigma\gamma\rho\rangle$, that have the same length and where the substring γ occurs in exactly the same position, then the method will give, as extracted genes, also their chimeric combinations $\langle\sigma\gamma\psi\rangle$ and $\langle\varphi\gamma\rho\rangle$. In other words,

Table 6
A DNA extraction algorithm

$XPCR - Extract(P,\gamma)=$

1. $S := \emptyset$;
2. $L := length(P)$;
3. **for each** $n \in L$ **do**
4. $R_1 := \emptyset$, $R_2 := \emptyset$, $Q := \emptyset$, $P_1 := \emptyset$, $P_2 := \emptyset$;
5. **begin**
6. $P := separate(P,n)$;
7. $P := infix(P,\alpha,\beta)$;
8. $(P_1,P_2) := split(P)$;
9. $P_1 := PCR(P_1,\alpha,\bar{\gamma})$;
10. **for each** $m < n$ **do** $R_1 := mix(R_1, separate(P_1,m))$;
11. $P_2 := PCR(P_2,\gamma,\bar{\beta})$
12. **for each** $m < n$ **do** $R_2 := mix(R_2, separate(P_2,m))$;
13. $Q := mix(R_1,R_2)$;
14. $Q := PCR(Q,\alpha,\bar{\beta})$;
15. $Q := separate(Q, n + |\alpha| + |\beta|)$;
16. $S := mix(S,Q)$;
17. **end**
18. **output** S.

Table 7
A DNA extraction algorithm avoiding chimers

$XPCR - PureExtract(P,\gamma)=$

1. **input** P;
2. $L := length(P)$;
3. **for each** $n \in L$ **do**
4. **begin**
5. $P := separate(P,n)$;
6. $(P,Q) := split(P)$;
7. $P := infix(P,\alpha,\lambda)$;
8. $Q := infix(Q,\lambda,\beta)$;
9. $P := PCR(P,\alpha,\overline{\gamma\delta})$;
10. $P := mix(P,Q)$;
11. $P := PCR(P,\alpha,\bar{\beta})$;
12. $P := separate(P, n + |\alpha| + |\beta|)$;
13. **output** P;
14. **end**.

if we define $Recombine(L,\gamma) = L + \{\langle\alpha\gamma\beta\rangle, \langle\eta\gamma\delta\rangle \,|\, \langle\alpha\gamma\delta\rangle, \langle\eta\gamma\beta\rangle \in L\}$, then $Extract(P,\gamma)$ coincides with $Recombine(Type(P),\gamma)$. For this case a further check is necessary, and it could be done for example as follows. When the extracted genes are sequenced, if a sequence before γ and a sequence after γ are chosen, then a PCR with them as primers, applied to a copy of the initial pool, will produce an exponential amplification of only those sequences which belong to a gene of the initial pool.

The extraction algorithm in Table 7, where δ is any *sufficiently long* string that does not occur in the pool P after γ, has a more sophisticated logic; differently from the previous extraction algorithm, it avoids the production of chimeric forms. Its correctness follows from the PCR lemma, while a graphical description of it (for any length of strings in the given pool) is displayed both in Table 8 and in Fig. 6.

As a consequence of the PCR Lemma, the last PCR of the XPCR-PureExtract Algorithm does not (exponentially) amplify double strings such as those on the right side of Fig. 6, which would generate chimers. On the contrary, double strings on the left side of Fig. 6 are (exponentially) amplified (after an extension) by the algorithm, and they generate the γ-superstrings of the initial pool.

Proposition 1 (Pure Extract Method Correctness). *Pure Extract-XPCR*(P,γ) *algorithm extracts only the* γ*-superstrings of* P (*that is, with no production of new* γ*-superstrings which do not belong to* P).

Proof. Consider 'genes' with a given length, they are split in two pools P, Q. The pool P is elongated with α and the pool Q with β. If $PCR(P,\alpha,\overline{\gamma\delta})$ is applied to the pool P after elongation, a string $\langle\alpha\eta\gamma\delta\rangle$ is amplified coming from genes including γ to which δ is added after γ. Let us call again P the pool generated by this PCR. When P is mixed to Q and again we call P the resulting pool, then $PCR(P,\alpha,\bar{\beta})$, due to the presence of δ, will provide Y forms. To these forms, according to PCR lemma, an exponential amplification applies, when a strand s_1 of type $\alpha\eta\gamma\delta$ of P hybridizes with a strand s_2 of type $\frac{\lambda}{(\theta\gamma\zeta\beta)^c}$ of Q, but this happens only if $\eta = \theta$, that is, if s_1 and s_2, apart the

Table 8
The computation flow of XPCR-PureExtract algorithm

$T = \{\xi_1, \ldots, \xi_n\}, \gamma$	
$\Downarrow split(T)$	
$P := \{\xi_1, \ldots, \xi_n\}$	$Q := \{\xi_1, \ldots, \xi_n\}$
$\Downarrow infix(P, \alpha, \lambda)$	$\Downarrow infix\ (Q, \lambda, \beta)$
$P := \{\alpha\xi_1, \ldots, \alpha\xi_n\}$	$Q := \{\xi_1\beta, \ldots, \xi_n\beta\}$
$\Downarrow\ PCR(P, \alpha, \overline{\gamma\delta})$	|
	|
$P := \{\alpha\eta\gamma\delta \mid \eta\gamma\zeta \in T\}$	|
$\Downarrow R := mix(P, Q)$	
$R := \{\xi_1\beta, \ldots, \xi_n\beta,\ \alpha\eta\gamma\delta \mid \eta\gamma\zeta \in T\}$	
$\Downarrow\ R := PCR(R, \alpha, \bar{\beta})$	
$R := \{\alpha\eta\gamma\zeta\beta \mid \eta\gamma\zeta \in T\}$	

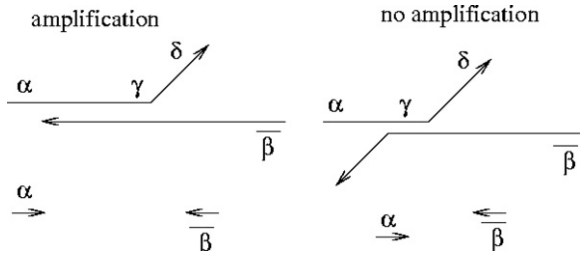

Fig. 6. Forms generated during the last step of the for cycle in the Pure Extract algorithm.

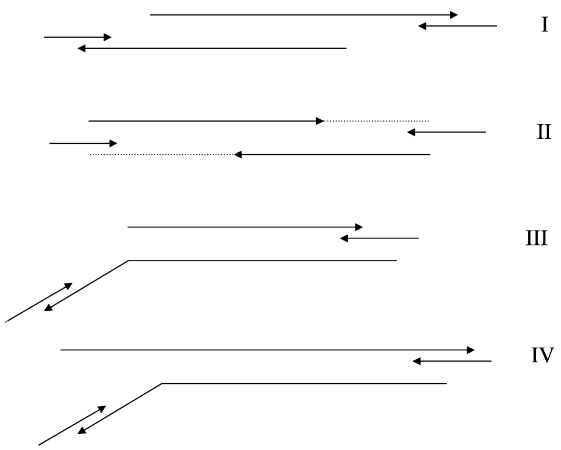

Fig. 7. Four cases of PCR corresponding to the four cases of PCR Lemma.

elongations, are pieces of genes of the same type. In fact, in the case that these pieces come from different genes, the part η of s_1 does not hybridize with the corresponding part $\bar{\theta}$ of s_2, so that a YY form arises with primers possibly put as in the case illustrated in Fig. 6, where no exponential amplification is possible. In conclusion, as overall effect of the procedure, only genes including γ will be elongated with both strings α and β and so are separable by length from the pool. □

5. DNA recombination by XPCR

Let P a DNA pool of type $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and Q a pool of type $\{\beta_1, \beta_2, \ldots, \beta_n\}$. The problem of generating all possible recombinations of pools P and Q is that of obtaining a pool of type $L = \{\eta_1\eta_2 \ldots \eta_n | \eta_1 \in \{\alpha_1, \beta_1\}, \eta_2 \in \{\alpha_2, \beta_2\} \ldots \eta_n \in \{\alpha_n, \beta_n\}\}$. Of course L contains 2^n different strings; we call it the n dimensional complete recombination of P and Q. In DNA computing this is an important step for encoding all the possible solutions of a combinatorial problem. For example, if α_i, β_i encode the two possible values of a Boolean variable x_i, then any string of L usually encodes a possible solution of the problem. True solutions are obtained by generating L and then by extracting strings which satisfy the requirements of the problem. But, apart this DNA computing interest, DNA recombination has an important biological meaning. For example, in the immunological system recombination is a key feature for generating the antibody repertory which is essential to the security system preserving the biological identity. However, in general, DNA pools generated by XPCR recombination could be very useful in the analysis of many aspects related to DNA hypervariability.

Let us present a DNA recombination method based on XPCR [9]. We show that a simple and efficient algorithm based on XPCR can provide the n dimensional complete recombination of P and Q. The main intuition behind our recombination method was inspired by Braich et al. [3], where only particular sequences were studied to avoid mismatch phenomena. Let us consider the following four *initial sequences*, where n is an odd number (if n is even, the roles of α_n and β_n are inverted in the last two sequences):

Positive: $\eta_1 = \langle \alpha_1\alpha_2\alpha_3\alpha_4 \ldots \alpha_n \rangle$
Negative: $\eta_2 = \langle \beta_1\beta_2\beta_3\beta_4 \ldots \beta_n \rangle$
Positive–Negative: $\eta_3 = \langle \alpha_1\beta_2\alpha_3\beta_4 \ldots \alpha_n \rangle$
Negative–Positive: $\eta_4 = \langle \beta_1\alpha_2\beta_3\alpha_4 \ldots \beta_n \rangle$.

Let us call α-string any element of $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and β-string any element of $\{\beta_1, \beta_2, \ldots, \beta_n\}$. Type L is the set of all the possible ordered combinations of n α-strings and β-strings. We call XPCR rule r_γ, and write it as $\xi_1,\ \xi_2 \xrightarrow{r_\gamma} \zeta$, the relation between strings ξ_1, ξ_2, ζ that holds when $\xi_1 = \langle \alpha\delta\gamma \ldots \rangle$, $\xi_2 = \langle \ldots \gamma\theta\beta \rangle$, and $\zeta = \langle \alpha\delta\gamma\theta\beta \rangle$. Any string that is a combination of α-strings and β-strings can be obtained from $\eta_1, \eta_2, \eta_3, \eta_4$ by suitable XPCR rules r_γ where γ is an α-string or a β-string. For example, with $n = 7$, the string $\alpha_1\alpha_2\beta_3\alpha_4\beta_5\beta_6\beta_7$, can be obtained in the following way:

$$\eta_1,\ \eta_4 \xrightarrow{r_{\alpha_2}} \alpha_1\alpha_2\beta_3\alpha_4\beta_5\alpha_6\beta_7, \quad \eta_2 \xrightarrow{r_{\beta_5}} \alpha_1\alpha_2\beta_3\alpha_4\beta_5\beta_6\beta_7.$$

It can be easily shown that the order of application of the rules is not relevant because the same string can be also obtained by permuting the order of application of the rules (from the same initial strings).

Table 9
The quaternary recombination algorithm

$XPCR - Recombination(\{\alpha_1, \alpha_2, \ldots, \alpha_n\}, \{\beta_1, \beta_2, \ldots \beta_n\})=$
let $Type(P) = \{\eta_1,\eta_2,\eta_3,\eta_4\}$;
$P := infix(P, \alpha, \beta)$;
for $i = 2, n-1$ **do**
begin
$P := PCR(P, \alpha, \bar{\alpha}_i)$;
$P := PCR(P, \alpha_i, \bar{\beta})$;
$P := PCR(P, \alpha, \bar{\beta}_i)$;
$P := PCR(P, \beta_i, \bar{\beta})$;
$P := PCR(P, \alpha, \bar{\beta})$;
end;
output P.

Let us consider the set of XPCR rules $R = \{r_{\alpha_2}, r_{\alpha_3}, \ldots, r_{\alpha_{n-1}}, r_{\beta_2}, r_{\beta_3}, \ldots, r_{\beta_{n-1}}\}$. A 'quaternary XPCR recombination' which produces the type L from $\{\eta_1, \eta_2, \eta_3, \eta_4\}$ is effectively specified by the algorithm displayed in Table 9. A completeness claim of our quaternary XPCR recombination method is given by the following proposition which is an easy consequence of XPCR definition and of the particular structure of the pool to which we apply this recombination method [9].

Proposition 2 (Recombination Method Correctness). *If all the XPCR rules of R are sequentially applied to a DNA pool of Type* $\{\eta_1, \eta_2, \eta_3, \eta_4\}$, *then a final DNA pool is obtained of Type* $L = \{\xi_1\xi_2 \ldots \xi_n \mid \xi_1 \in \{\alpha_1, \beta_1\}, \xi_2 \in \{\alpha_2, \beta_2\} \ldots \xi_n \in \{\alpha_n, \beta_n\}\}$.

The quaternary recombination algorithm has the additional advantage to be equipped by a couple of special strings, called *recombination witnesses* [9], such that, if they are present in the final pool then the whole library L is present too. This information guarantees that no experimental error has occurred during the implementation of the algorithm. A combinatorial analysis of the mechanism underlying this property has been developed in [6].

6. Conclusion

The paper showed that concepts rooted in 'Formal Language Theory' are capable of a surprising explicative power with respect to basic biological phenomena. In principle, this is not so strange, being DNA molecules special words of a four-symbol alphabet, but the passage from this intuition to a rigorous mathematical description is not so evident. We showed that this is possible by means of a symbolic notation which translates recombination mechanisms into appropriate mathematical operations. We demonstrated that a mathematical abstraction of DNA recombination may highlight novel, unexpected, properties of PCR leading to the discovery of XPCR based methodologies in different contexts, such as DNA extraction and recombination. Our algorithms were validated by laboratory experiments, to verify that some transformations are performed according to some combinatorial schemata [7,9]. In this context, computations are not only pure theoretical constructs, but somewhat with a physical content, testable like mathematical equations describing motions. DNA

test tubes are the reality where algorithms on multisets of double strings become real processes obeying to specific rules, that are intrinsically related to a special kind of discrete operations.

References

[1] L.M. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021.
[2] K. Benenson, T. Paz-Elitzur, R. Adar, E. Keinan, Z. Livneh, E. Shapiro, Programmable and autonomous computing machine made of biomolecules, Nature 414 (2001) 430.
[3] R.S. Braich, C. Johnson, P.W.K. Rothemund, D. Hwang, N. Chelyapov, L. Adleman, Solution of a satisfiability problem on a gel-based DNA computer, in: A. Condon, G. Rozenberg, (Eds.), DNA Computing, 6th International Workshop on DNA-based Computers, Leiden, Netherlands, Lecture Notes in Computer Science 2054, Springer, 2000, p. 27.
[4] R.S. Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund, L. Adleman, Solution of a 20-variable 3-SAT problem on a DNA computer, Science 296 (2002) 417.
[5] C. Ferretti, G. Mauri, C. Zandron (Eds.), Proceedings of 10th International Meeting on DNA Based Computers: DNA 10, Milan, Italy, June 2004, Revised Selected Papers, Lecture Notes in Computer Science 3384, Springer, 2005.
[6] G. Franco, Combinatorial properties of DNA libraries generated via null context splicing rules, submitted for publication.
[7] G. Franco, C. Giagulli, C. Laudanna, V. Manca, DNA Extraction by cross pairing PCR, in: 5, p. 106.
[8] G. Franco, V. Manca, An algorithmic analysis of DNA structure, Soft Comput. 9 (2005) 761.
[9] G. Franco, V. Manca, C. Giagulli, C. Laudanna, DNA recombination by XPCR, in: A. Carbone, N.A. Pierce (Eds.), Proceedings of the 11th International Workshop on DNA Computing: DNA 11, London, Ont., Canada, June 2005, Revised Selected Papers, Lecture Notes in Computer Science 3892, Springer, 2006, p. 55.
[10] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, Bull. Math. Biol. 49 (1987) 737.
[11] J.Y. Lee, H.W. Lim, S.-I. Yoo, B.T. Zhang, T.H. Park, Efficient initial pool generation for weighted graph problems using parallel overlap assembly, in: 5, p. 215.
[12] R.J. Lipton, DNA solutions of hard computational problems, Science 268 (1995) 542.
[13] V. Manca, On the logic and geometry of bilinear forms, Fundam. Inform. 64 (2005) 261.
[14] V. Manca, C. Zandron, A clause string DNA algorithm for SAT, in: N. Jonoska, N.C. Seeman (Eds.), DNA Computing. 7th International Workshop on DNA-Based Computers, DNA 7. Tampa, FL, USA, June 2001. Revised Papers, Lecture Notes in Computer Science 2340, Springer, 2002, p. 172.
[15] S.H. Park, H. Yan, J.H. Reif, T.H. LaBean, G. Finkelstein, Electronic nanostructures templated on self-assembled DNA scaffolds, Nanotechnology 15 (2004) S525.
[16] Gh. Păun, G. Rozenberg, A. Salomaa, DNA Computing. New Computing Paradigms, Springer, Berlin, 1998.
[17] J.A. Rose, M. Hagiya, R.J. Deaton, A. Suyama, A DNA-based in vitro genetic program, J. Biol. Phys. 28 (2002) 493.
[18] G. Rozenberg, A. Salomaa, Handbook of Formal Languages, vol. 3, Springer, Berlin, 1997.
[19] R.K. Saiki, S. Scharf, F. Faloona, K.B. Mullis, G.T. Horn, H.A. Erlich, N. Arnheim, Enzymatic amplification of β-globin genomic sequences and restriction site analysis for diagnosis of sickle cell anemia, Science 230 (1985) 1350.
[20] B. Schu, PCR leaves its teen years, and lingering questions, behind, Genomics Proteomics 5–1 (2005) 20.
[21] N.C. Seeman, DNA in a material worldblocking, Nature 421 (2003) 427.
[22] D. Soloveichik, E. Winfree, Complexity of self-assembled shapes, in: 5, p. 344.