A universal functional approach to DNA computing and its experimental practicability

Thomas Hinze Monika Sturm

Dresden University of Technology, Dept. of Theoretical Computer Science, Germany
e-mail: {sturm,hinze}@tcs.inf.tu-dresden.de
www: http://wwwtcs.inf.tu-dresden.de/dnacomp

Abstract. The rapid developments in the field of DNA computing reflects two substantial questions: 1. Which models for DNA based computation are really universal? 2. Which model fulfills the requirements to a universal lab-practicable programmable DNA computer that is based on one of these models? This paper introduces the functional model DNA-Haskell focussing its lab-practicability. This aim could be reached by specifying the DNA based operations in accordiance to an analysis of molecular biological processes. The specification is determined by an abstraction level that includes nucleotides and strand end labels like 5'-phosphate. Our model is able to describe DNA algorithms for any NP-complete problem – here exemplified by the knapsack problem – as well as it is able to simulate some established mathematical models for computation. We point out the splicing operation as an example. The computational completeness of DNA-Haskell can be supposed. This paper is based on discussions about the potenzial and limits of DNA computing, in particular the practicability of a universal DNA computer.

1 Introduction

Many research activities during the last years led to concepts of alternative hardware platforms. Quantum computing, neural computing and molecular computing are considered as components of the so called future computing. Molecular computing distinguishes three kinds of molecules serving as data structures: DNA, RNA, and proteins. Particularly the DNA computing is on the way to become a practicable tool for an efficient handling of algorithmic problems whose solution using traditional computers seems to be too time consuming.

In parallel a variety of models for DNA computing was developed and discussed under different aspects. The objectives were their power, their complexity, their stability, and last but not least their practicability. The DNA computer represents a powerful tool to describe, execute and implement parallel processes. According to the architecture classification of Flynn, DNA computing falls into the category of Single Instruction Multiple Data (SIMD). The massive data parallelism embodies the potential of DNA computing.

It is well known that DNA operations can cause side effects in a way that the results of algorithms does not fit to the expectation. Typical side effects are unwanted DNA strands inside a final tube, loss of wanted DNA strands and artifacts, that means DNA strands of an unwanted nucleotide sequence or structure. The clue to handle side effects in DNA computing can consist in the idea to include these effects into the definition of the DNA operations as far as possible.

Here, we provide an extended functional programming language, DNA-Haskell which allows the description and simulations of algorithms directly based on the low level behavior of recombinant DNA techniques. This proves the suitability of this language to simulate the solution of NP-problems by laboratory experiments. A laboratory implementation of DNA-Haskell solved the knapsack problem with three objects.

2 DNA-Haskell

The functional language DNA-Haskell was conceived on the base of the functional language Haskell. The decision for a functional language is due to the fact that the abstraction level of a functional program is close to the level of problem specification. This quality allows an easier mathematical handling. Only because of this property correctness proofs, verification, and analysis of program properties are possible. DNA computing as a modern model for computations is convincing only if a formal description of the labwork can be done.

There are two possibilities to implement algorithms in DNA-Haskell. On one hand all functions can be performed massive data parallel in the laboratory. So, the computer science utilizes molecular biological techniques to solve a certain class of algorithmic problems. The laboratory is considered as biohardware, a special SIMD architecture. On the other hand DNA-Haskell can be implemented as a simulation of molecular biological processes on a personal computer with SISD architecture (Single Instruction Single Data). The simulation does not use parallel data processing, but it supports the preparation of lab-experiments and the development of DNA based algorithms including side effects.

Every DNA strand is described as linearized finite sequence of nucleotides and/or nucleotide pairs. Every strand end (5' and 3') is labeled by a chemical group from table 1. This allows to distinguish between DNA single strands and DNA double strands, to evaluate the quality of annealings, to localize restriction sites and to control operations like ligation by appropriate strand end labelings.

A tube is considered in DNA-Haskell as a finite set of different DNA strands. Trivially, a tube can be empty and contains no DNA strand. During handling with DNA in the laboratory many duplicates of identical DNA strands exist in one tube inducing a large redundancy. DNA-Haskell abstracts from this redundancy and represents several identical DNA strands in one tube by one instance (non-restrictive model). Two DNA strands are identical if they are congruent in their nucleotide/nucleotide pair sequence and all labels at each position.

Declaration of DNA data types

A DNA strand is described as a list composed by basepairs and two labelpairs on the ends. The symbol * encodes a single strand segment. A tube is considered as a list of DNA strands. The figure 1 describes some examples.

DNA-Haskell symbol	label (molecule or chemical group)
@	hydroxyl group (unlabeled)
P	phosphate group
B	Biotin
C	Cy5

Table 1. considered strand end labels

```
data Base      = A | T | C | G | *
data Label     = P | B | C | @
data Basepair  = [A,T]|[T,A]|[C,G]|[G,C]|[A,*]|[T,*]|[C,*]|[G,*]|
                 [*,T]|[*,A]|[*,G]|[*,C]
data Labelpair = [@,@]|[@,P]|[@,B]|[@,C]|[P,@]|[B,@]|[C,@]|[P,P]|[B,B]...
data Strand    = NIL  | Cons Basepair Strand
data Dnastrand = Labelpair ++ Strand ++ Labelpair
data Tube      = NIL  | Cons Dnastrand Tube
```

	DNA strand	Notation in DNA-HASKELL	for example
single strands unlabeled	5'-ATCGAT-3'	[[@,@]] ++ [[A,*],[T,*],[C,*],[G,*],[A,*],[T,*]] ++ [[@,@]]	ATCGAT
	3'-TAGCTA-5'	[[@,@]] ++ [[*,T],[*,A],[*,G],[*,C],[*,T],[*,A]] ++ [[@,@]]	
single strands labeled	5'P-ATCGAT-3'B	[[P,@]] ++ [[A,*],[T,*],[C,*],[G,*],[A,*],[T,*]] ++ [[B,@]]	ATCGAT
	3'B-TAGCTA-5'P	[[@,B]] ++ [[*,T],[*,A],[*,G],[*,C],[*,T],[*,A]] ++ [[@,P]]	
double strands unlabeled	5'-ATCGCA-3' 3'-TAGCGT-5'	[[@,@]] ++ [[A,T],[T,A],[C,G],[G,C],[C,G],[A,T]] ++ [[@,@]]	ATCGCA TAGCGT
	5'-TGCGAT-3' 3'-ACGCTA-5'	[[@,@]] ++ [[T,A],[G,C],[C,G],[G,C],[A,T],[T,A]] ++ [[@,@]]	
double strands labeled	5'P-ATCGCA-3' 3'-TAGCGT-5'B	[[P,@]] ++ [[A,T],[T,A],[C,G],[G,C],[C,G],[A,T]] ++ [[@,B]]	ATCGCA TAGCGT
	5'B-TGCGAT-3' 3'-ACGCTA-5'P	[[B,@]] ++ [[T,A],[G,C],[C,G],[G,C],[A,T],[T,A]] ++ [[@,P]]	

Fig. 1. representation of DNA strands

Selected DNA operations

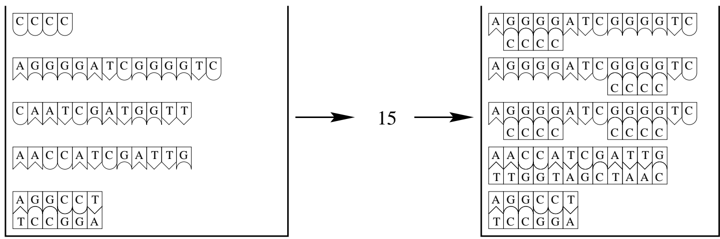
DNA strands can be recombined by exchange of nucleotide sequences and labels. The recombinant technology uses a variety of molecular biological operations that are in part considered in DNA computing. These operations are introduced by their effects adapted from the biochemistry and by their typing.

- Synthesis

  The function `syn :: [Char] -> Tube` creates a DNA single strand from the input base sequence pretended in 5'-3'-direction. Both ends are unlabeled.

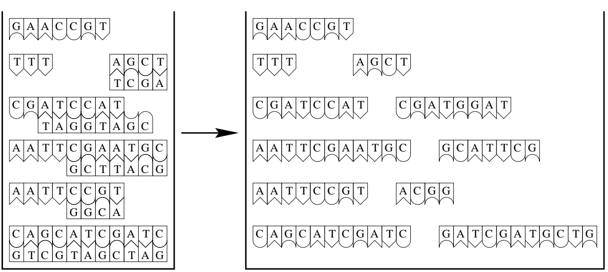
- Annealing

  The function `ann :: Tube -> Int -> Tube` simulates the biological operation annealing. Single strands and double strands with sticky ends anneal to each other only when they are complementary. If so, they are forming double strands. All combinations are generated. The simpliest form is the annealing of two nucleotides that results in a base pair. The function needs an integer number that limits the length of the new annealed strands.



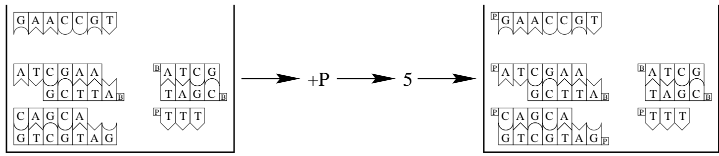
– Melting

The function `mel :: Tube -> Tube` is related to the function annealing because it represents the inversion of the biological annealing effect. Single strands will not be changed by melting. Double strands are separated into their single strand counterparts. After executing this operation the tube will contain only single strands.



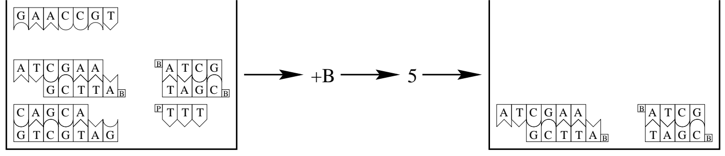
– Labeling

The function `lab :: Tube -> [Char] -> Int -> Tube` sets or removes certain labels on strand ends. The string characterizes the label and the integer number defines which strand end should be labeled (3' or 5'). Possible labels are +/- P (Phosphorylating/ Dephosphorylating), +/- B (Biotinylating/ Debiotinylating) und +/- C (set Cy5-label/ remove Cy5-label).



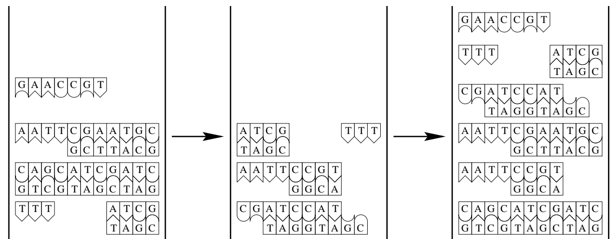
– SepLabel

The function `spl :: Tube -> [Char] -> Int -> Tube` selects exactly those strands from the input tube into the output tube that carry (+) or not (-) a certain label. The integer number defines whether the separation is performed at the 3'-ends (3) or at the 5'-ends (5). The operation effects single strands as well as double strands.



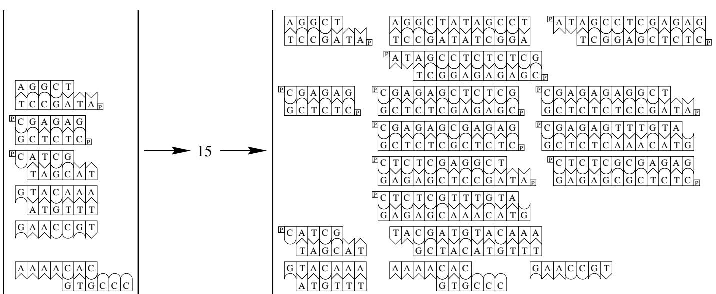
– Union

The function `un :: Tube -> Tube -> Tube` combines the components of either tubes. Identical strands are taken only once.



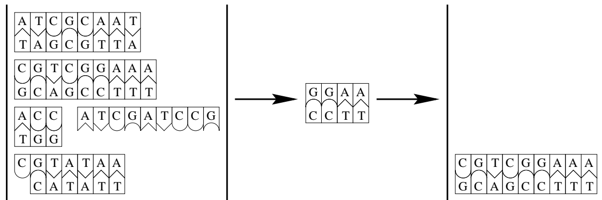
– Ligation

The function `lig :: Tube -> Int -> Tube` simulates the biological ligation. All double strands inside the tube can be linked to itself or to another double strand under following conditions: The strands have compatible complementary ends and at least one of the connected strands has to be modified by 5'-phosphorylation. The generation of new concatenated strands will continue until the defined maximum in length is reached.



– Extraction

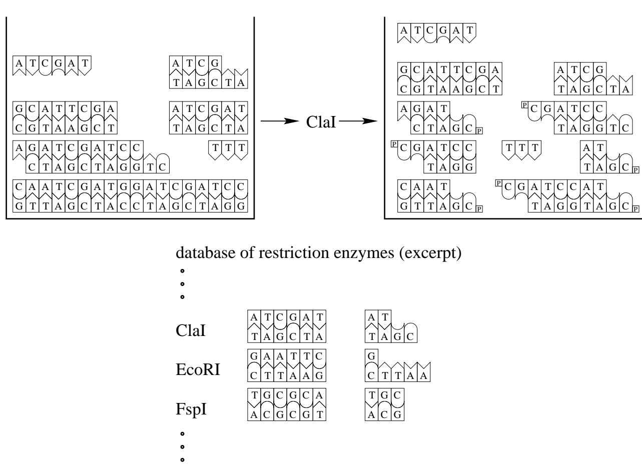
An application of the technique affinity purification is modelled by the function `extr :: Tube -> Dnastrand -> Tube`. All strands carrying the sequence given as `Dnastrand` are separated into the output tube. This allows a separation by subsequence.



– Cut

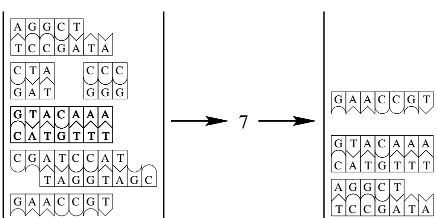
The function `cut :: Tube -> [Char] -> Tube` simulates the effect of different restriction enzymes and cleaves all appropriate strands. A database contains information about the names of available restriction enzymes, their sequences and restriction sites. The effects of the different restriction enzymes are defined

by their restriction sides. Single strands are not able to be cleaved. All resulting ends of the strands are modified by 5'-phosphorylation.



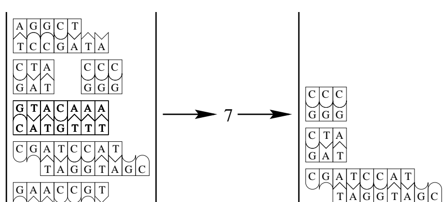
– CuttingOut

The function `cout :: Tube -> Int -> Tube` selects all strands with the defined length into the resulting tube.



– FilterLength

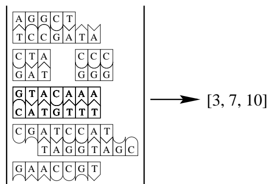
The function `fltl :: Tube -> Int -> Tube` filters all strands that vary from the defined length. These strands are collected in the resulting tube.



– Electrophoresis

The electrophoresis is formalized by the typing `elt :: Tube -> [Int]`. It sorts

all strands of one tube according to length. The function returns a list with all appropriate lengths.



3 The solution of an integer knapsack problem

The integer knapsack problem belongs to the class of NP-complete problems. There are n objects with the weights $a_i \in \mathcal{N}, \quad 1 \leq i \leq n$ and a number $b \in \mathcal{N}$. The integer knapsack problem is based on the decision whether or not a subset $I \subseteq \{1, 2, \ldots, n\}$ exists under the condition $\sum_{i \in I} a_i = b$. The parameter n represents the problem size. n objects allow 2^n possibilities to pack the knapsack. For example, we use the integer knapsack problem with three objects, their weights $a_1 = 719, a_2 = 393, a_3 = 270$ and $b = 1112$.

The object weights are encoded by DNA double strands with lengths according to the weights. The plasmid pQE30 forms the basic material. It is a ring-shaped double stranded DNA and can be isolated from bacteria in the laboratory. pQE30 consists of 3462 base pairs (bp) with well-known sequence.

For extracting DNA fragments with 719bp, 393bp and 270bp from the plasmid it is cleaved by the restriction enzymes PvuII and HinP1I and the resulting fragments are modified by 5'-dephosphorylation. PvuII is a blunt end cutter. Its restriction site appears only once in pQE30. The restriction site of HinP1I – a sticky end cutter – exists on several different positions in pQE30. A subset of DNA is not processed. The strands are separated by length through an agarose gel electrophoresis, see figure 2A (lane1: cleaved and dephosphorylated DNA fragments. A subset represents the object weights; lane2: 50bp marker).

The bands with 719bp, 393bp, and 270bp are excised from the gel and the DNA is extracted. The following step produces all possibilities to pack the knapsack using the objects with the weights a_1 and a_2. That means that DNA fragments with the lengths 0bp, 719bp, 393bp, and 719+393bp should be created. The ligation concatenates end compatible DNA double strands. The condition for concatenating is that at least one of the two strands that is to be ligated is phosphorylated at the 5' end. For this purpose, an aliquot from the tube with the 719bp fragments is phosphorylated at the 5' end. After that, the ligation runs with the dephosphorylated 719bp fragments, the 5' phosphorylated 719bp fragments and the dephosphorylated 393bp fragments. This ensures that only one 393bp fragment can be joint to one 719bp fragment to avoid the generation of strands with more than two fragments. Fragments with 719+719bp, 719+719+393bp and longer can also appear but they are outside the considered interval [6].

Using the third object increases the number of possibilities to pack the knapsack from 4 to 8. To do so, an aliquot of the ligation product is 5' phosphorylated. Subsequently, the unlabeled ligation product, the 5' phosphorylated product and the unlabeled 270bp fragments form the input for a new ligation. This ligation produces DNA strands with the lengths representing the knapsack's weigths of all

packing possibilities. The bands are visualized by agarose gel electrophoresis, see figure 2B (lanes1, 2: possible knapsack weights using 3 objects except the empty knapsack; lane3: 100bp marker).

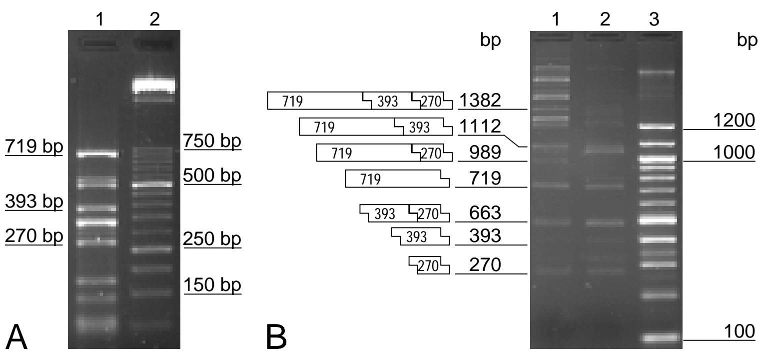


Fig. 2. agarose gel photos. A: encoded object weights, B: knapsack weights

The band of the length 1112bp could be identified answering "yes" as problem solution. The whole process in the laboratory needed nearly 40 hours. The algorithm is scalable from 3 to n objects. The number of operations increases linearly in the number of available objects. Figure 3 describes the algorithm for 2 objects.

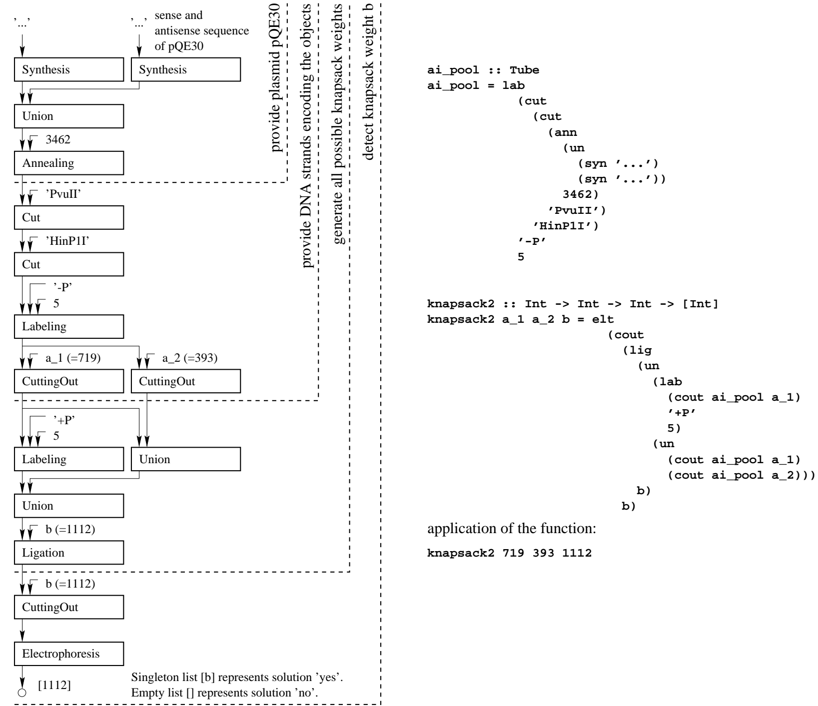


Fig. 3. knapsack algorithm as a flowchart (left) and using the DNA-Haskell syntax (right)

4 Simulating the splicing operation using DNA-Haskell functions

The splicing operation forms the core of all types of splicing systems and embodies an abstract formal emulation of DNA recombinant techniques cut with restriction enzymes (digestion) and ligation [2]. It is based on elements of mostly infinite sets that express DNA strands, further named words of formal languages. The description of the splicing operation on words of formal languages also leads to a generalization of the effect that is caused by digestion and ligation. The generalization suppresses certain DNA strands resp. words that can really additional occur during the ligation process as side effects. Here, we propose a sequence of DNA-Haskell operations that simulate the splicing operation on linear data structures defined by a splicing rule in [3].

Consider an alphabet Σ, and two symbols \$ and # not in Σ. A splicing rule over Σ is a string $r = \alpha_1 \# \beta_1 \$ \alpha_2 \# \beta_2$, where $\alpha_i, \beta_i \in \Sigma^*, \quad 1 \leq i \leq 2$. For each such rule r and strings $x, y, w, z \in \Sigma^*$ we define

$$(x, y) \vdash_r (z, w) \text{ if and only if } \begin{array}{ll} x = x_1\alpha_1\beta_1 x_2, & y = y_1\alpha_2\beta_2 y_2, \\ z = x_1\alpha_1\beta_2 y_2, & w = y_1\alpha_2\beta_1 x_2. \end{array}$$

A wet splicing system and its experimental implementation was introduced in [5]. The results of this approach encourage the assumption that the splicing operation can be performed practically. This fact represents the first step to establish a universal DNA computer based on splicing.

Unfortunally the ligation produces unwanted DNA fragments of the composition $x_1\alpha_1\alpha_2 y_1$ and $y_2\beta_2\beta_1 x_2$. These DNA fragments occur because of the compatibility of the sticky ends resulting from the digestions $\alpha_1|\beta_1$ and $\alpha_2|\beta_2$. These additional unwanted DNA fragments can hardly influence the final result of iterated executions of splicing operations, often used to generate a formal language by applying splicing rules of a splicing system. The following figure 4 proposes an idea how to overcome this insufficiency. Let x and y encoded by DNA double strands.

We are able to describe the splicing operation in an experimental convincing way. DNA computing should lead to an unconventional universal model for computations. Splicing systems based on the splicing operation represent an exact mathematical model according to this aim [4]. A practical execution of different splicing systems can contain the scenario from figure 4 as the central part. The challenge consists in finding a way how to adapt the real molecular biological processes in the laboratory to the formal definition of the splicing operation. The focus lies in the laboratory-like modelling of splicing systems to generate regular, context free, and recursive enumerable languages, so calles type 0 languages. Recent studies deal with this topic.

5 Conclusions

DNA-Haskell represents a model for DNA computing whose operations were implemented in the laboratory and contributed to the successful solution of a NP-complete problem. Both, the description of NP-complete problem solving DNA algorithms and the simulation of computational complete universal models is possible with DNA-Haskell. This model is also able to include the description of another existing algorithmic implementations reducing some side effects because of its closeness to the laboratory, so to say, these effects belong to the definitions of

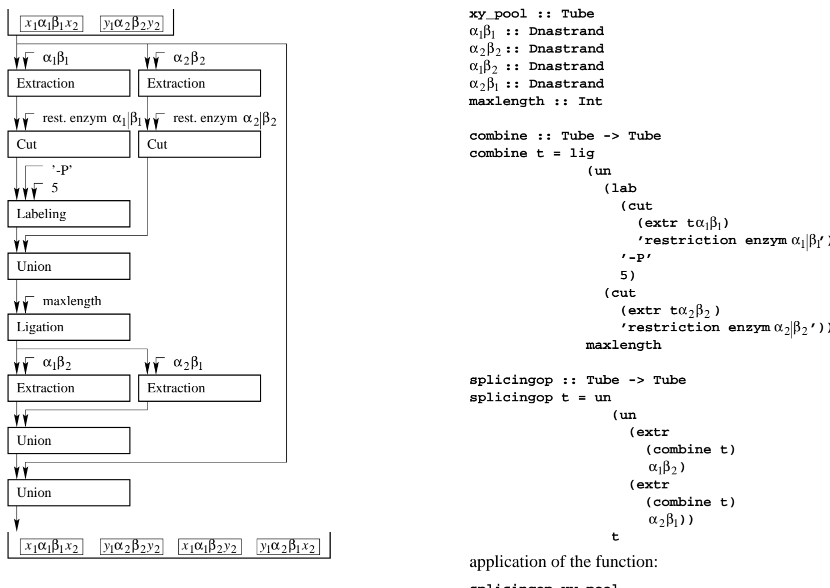


Fig. 4. splicing operation as a flowchart (left) and using the DNA-Haskell syntax (right)

the operations [1]. Beyond DNA-Haskell is suitable for description of established mathematical models for DNA computing. It fills the gap between models with a high abstraction level and practical implementations in the laboratory. The concept of DNA-Haskell arose by direct observations of molecular processes specifying the according functions and forming the operational semantics of DNA-Haskell. The computational completeness of DNA-Haskell can be assumed although it was not discussed in this paper.

Acknowledgements. This work is a result of the cooperation between the Department of Computer Science, Institute of Theoretical Computer Science, and the Faculty of Medicine Carl Gustav Carus, Department of Surgical Research, both Dresden University of Technology, Germany.

References

1. L.M. Adleman. Molecular computation of solutions to combinatorial problems. Science, vol. 266, p. 1021-1024, 1994
2. R. Freund, L. Kari, G. Paun. DNA computing based on splicing: the existence of universal computers. Theory of Computing Systems, vol. 32, p. 69-112, 1999
3. T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. Bulletin of the Mathematical Biology, 49(6), p. 737-759, 1987
4. L. Kari. DNA computing: the arrival of biological mathematics. The mathematical Intelligencer, vol. 19, 2, 1997
5. E. Laun, K. J. Reddy. Wet splicing systems. In Proceedings of the 3rd DIMACS Workshop on DNA Based Computers, University of Pennsylvania, p. 115-126, 1997
6. E. Stoschek, M. Sturm, T. Hinze. On a DNA experiment for solving a certain NP-complete problem. Technical Report TUD-FI99-02, Dresden University of Technology, 1999