A universal functional approach to DNA computing and its experimental practicability

Thomas Hinze • Monika Sturm
Institute of Theoretical Computer Science • Department of Computer Science
Dresden University of Technology

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? We introduce the functional model DNA-HASKELL focussing its lab-practicability. This aim could be reached by specifying the DNA based operations in accordance 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. The idea to this contribution is based on discussions about the potential and limits of DNA computing, in particular the practicability of a universal DNA computer.

Results

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 the operations. 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 by simulation of Turing machines and distributed splicing systems for recursive enumerable languages.

DNA-HASKELL

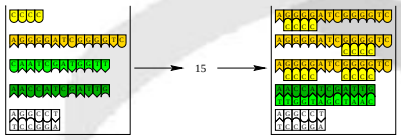
The functional language DNA-HASKELL was conceived on the base of the functional language HASKELL. The decision for a functional language is due to the fact that the abstraction level of a functional program is close to the level of problem specification. This quality allows an easier mathematical handling. 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.

Data structures

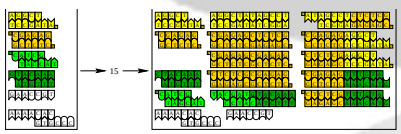
	DNA strand	denotation in DNA-HASKELL	example
single strands unlabeled	5'-ATCGAT-3'	[[@,@]] ++ [[A,*],[T,*],[C,*],[G,*],[A,*],[T,*]] ++ [[@,@]]	
	3'-TAGCTA-5'	[[@,@]] ++ [[*,T],[*,A],[*,G],[*,C],[*,T],[*,A]] ++ [[@,@]]	
single strands labeled	5'P-ATCGAT-3'B	[[P,@]] ++ [[A,*],[T,*],[C,*],[G,*],[A,*],[T,*]]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

Operations

- **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.



- **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.



- **Synthesis**, **Melting**, **Labeling**, **SepLabel**, **Union**, **Extraction**, **Cut**, **CuttingOut**, **FilterLength**, and **Electrophoresis** complete the set of operations. DNA algorithm simulations enable forecasts closed to the laboratory results, supporting low level design, test, and optimization of algorithmic lab-implementations.

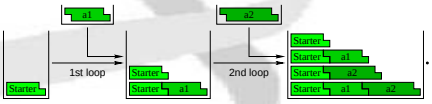
Knapsack problem

given: $a_1, \ldots, a_n \in \mathcal{N}$ and $b \in \mathcal{N}$.

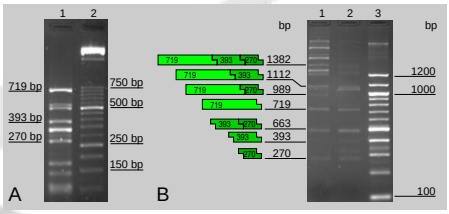
asked: Is there a subset $I \subseteq \{1, \ldots, n\}$ with $\sum_{i \in I} a_i = b$?

Lab-implemented example

We use a problem instance 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 was cleaved by PvuII and HinP1I. The resulting fragments were 5′-dephosphorylated and separated by agarose gel electrophoresis, see figure **A** (lane1: digestion product, lane2: 50bp marker). The bands with 719, 393, and 270bp were excised. The DNA was extracted into separate tubes representing weights.

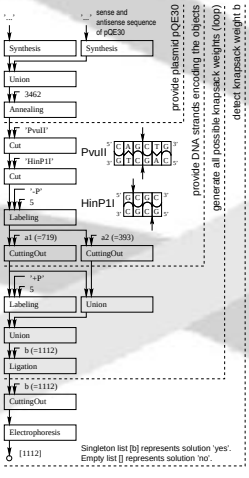


Agarose gel photos



A: encoded object weights, **B:** knapsack weights

Algorithm in DNA-HASKELL



The DNA algorithm for solution to the knapsack problem produces all nonempty combinations of the input DNA double strands (5′-dephosphorylated, end compatible) encoding a_i, $i = 1, \ldots, n$. Each combination contains every a_i at most once and represents a possible knapsack weight. Starting from a „Starter" fragment (one side end compatible, other side 5′-biotinylated), the combinations are generated by consecutively processing operational loops. Each loop embodies a split and combine strategy, adding a new a_i and doubling the number of combinations. The DNA pool is splitted into two halfs. One half is 5′-phosphorylated and merged with a new a_i and with the other half. A subsequent ligation produces new combinations. After including all a_i, a terminating fragment is added to all combinations. A subsequent PCR with starter and terminator sequences as primers extracts all valid combinations. Whether or not a combination of the length b occured can be determined by gel electrophoresis. Figure **B** shows the result of a simplified example without starter because of appropriate input strand lengths.

Splicing operation

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 $\alpha_i, \beta_i \in \Sigma^*$, $1 \leq i \leq 2$. For each such rule r and strings $x, y, w, z \in \Sigma^*$ we define

$$(x, y) \vdash_r (z, w) \text{ if and only if } \quad x = x_1\alpha_1\beta_1x_2, \quad y = y_1\alpha_2\beta_2y_2,$$
$$z = x_1\alpha_1\beta_2y_2, \quad w = y_1\alpha_2\beta_1x_2.$$

Splicing operation in DNA-HASKELL

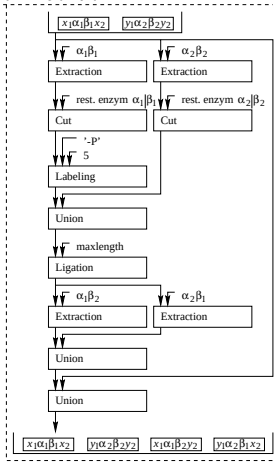
The splicing operation forms the core of all types of splicing systems and embodies an abstract formal emulation of DNA recombinant techniques cut with restriction enzymes (digestion) and ligation. It is based on elements of mostly infinite sets that express DNA strands, further named words of formal 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 as above (using functional DNA-HASKELL syntax and flowchart).

```
xy_pool :: Tube
α1β1 :: Dnastrand
α2β2 :: Dnastrand
α1β2 :: Dnastrand
α2β1 :: Dnastrand
maxlength :: Int
```

```
combine :: Tube -> Tube
combine t = lig
              (un
                (lab
                  (cut
                    (extr t α1β1)
                    'restriction enzym α1|β1')
                  '-P'
                  5)
                (cut
                  (extr t α2β2)
                  'restriction enzym α2|β2'))
              maxlength
```

```
splicingop :: Tube -> Tube
splicingop t = un
                 (un
                   (extr
                     (combine t)
                     α1β2)
                   (extr
                     (combine t)
                     α2β1))
                 t
```

application of the function: `splicingop xy_pool`



Contact:

Dresden University of Technology • Dept. of Computer Science • Inst. of Theoretical Computer Science • D-01062 Dresden • Germany
Dipl.-Inform. T. Hinze • e-mail: hinze@tcs.inf.tu-dresden.de
Dr. M. Sturm • e-mail: sturm@tcs.inf.tu-dresden.de
Internet: http://wwwtcs.inf.tu-dresden.de/dnacomp

DNA6 Sixth International Meeting on DNA Based Computers, Leiden, The Netherlands, 2000